手動テストのログを用いた有用な End-to-Endテストスクリプトの自動生成

切貫 弘之[1,a)]　丹野 治門[1,b)]

概要：テストの自動化は，市場の変化に迅速に対応して素早くソフトウェアをリリースするために重要である．しかし，画面の操作を伴うテストスクリプトの実装および保守のコストは自動テスト導入の大きな障壁となっている．一方，画面表示やユーザビリティのテストなど，現在の産業界におけるテストは人手に頼る部分も多く存在している．本研究では，このような人手によるテストのログを用いて，ページオブジェクトパターンを採用した有用なテストスクリプトを自動生成する手法を提案する．提案手法を利用することで，ソフトウェア開発で必要不可欠な手作業によるテストを行うだけで有用なテストスクリプトを生成することができ，テストスクリプトの実装・保守のコストを下げることが期待できる．

1. 背景

ソフトウェアテスト (以下，テスト) は，ソフトウェアの品質を評価し，改善するための重要な工程であり，ソフトウェア開発において多くのコストがかけられている [1], [2]．それゆえ，テストを効率化したりソフトウェアの欠陥をより多く発見したりすることを目的とした研究が多く行われている [3]．

テストの効率化のための重要な手法の 1 つとして，テスト自動化がある．テストの自動化は，回帰テストのように何度も繰り返し行われることが期待されるテストに対して特に効果を発揮する．近年，市場の変化に迅速に対応するためにソフトウェアのリリースサイクルを短くすることが求められており，そのためにもテストの自動化は重要である．

Web アプリケーション等の画面が存在するソフトウェアのテストでは，画面の操作を伴うユーザ目線のテストも必要である．このようなテストを End-to-End テストと呼ぶ．End-to-End テストの自動化のため，Selenium[4] に代表される Web ブラウザの操作を自動化するテストツールが広く用いられている．End-to-End テストの自動化を行うためには，テストで行う操作を再現するようなテストスクリプトの実装が必要である．また，テスト対象アプリケーションの修正に伴い，既存のテストスクリプトに修正が必要になる場合がある．Christophe らは，8 つの OSS に対して Selenium のテストスクリプトが記述されたソースコードの変更履歴を調査し，アプリケーションへのコミットが Selenium テストスクリプトにどの程度影響を与えるかを調べた．その結果，Selenium テストスクリプトの 75%は，アプリケーションへのコミット 9 回につき 1 回以上，期間では 2.05 日に 1 回以上変更されていることが分かった [5]．このように，Selenium テストスクリプトはアプリケーションの進化に伴い頻繁に更新されるため，その保守性は重要である．

End-to-End テストの自動化を行うための手法は，レコード＆リプレイツールを用いる手法とプログラミングによる手法の 2 種類に大別できる．レコード＆リプレイツールは，テスターが行った操作をスクリプトとして記録し，そのスクリプトを実行することで次回以降の動作検証を自動化するツールである．代表的なレコード＆リプレイツールとして，Selenium IDE がある．レコード＆リプレイツールを用いる手法はプログラミングの知識が無くても簡単にテストスクリプトが実装できる反面，テストスクリプトが単純な操作の羅列として出力されるため，保守性が低いという問題がある．一方，プログラミングによる手法では，Selenium webdriver 等の Web ブラウザを操作するライブラリを用いて，テストで行う操作をプログラムとして実装する．プログラミングによる手法は，開発者にスキルがあれば保守性の高いテストスクリプトを実装できる反面，実装に大きなコストがかかる問題がある．

Leotta らは，これらの 2 つの手法についての比較実験を行い，プログラミングによる手法はレコード＆リプレイ

[1] NTT ソフトウェアイノベーションセンタ
NTT Software Innovation Center, Minatoku, Tokyo 108-0023, Japan
[a)] hiroyuki.kirinuki.ad@hco.ntt.co.jp
[b)] haruto.tanno.bz@hco.ntt.co.jp

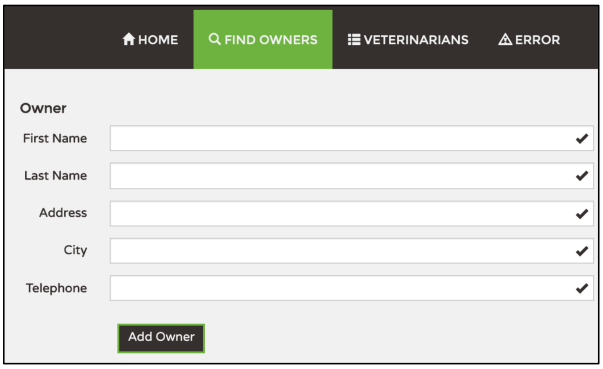
ツールを用いる手法に対して，テストスクリプトの実装にかかる時間が 32%–112%増える一方，修正にかかる時間を 16%–51%削減できることを示した [6]．また，実装と修正のトータルコストを考えると，ほとんどの場合，3 回の修正が必要になった時点でプログラミングによる手法の方がコストが低くなることを示した．

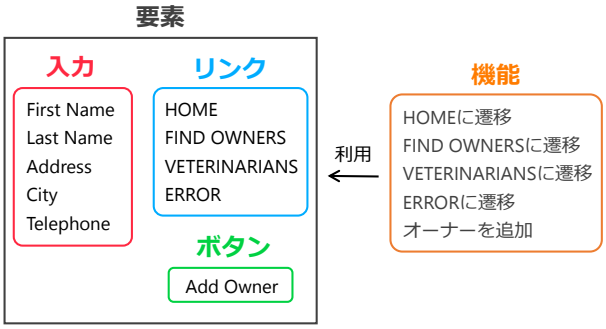
Leotta らの実験でプログラミングによる手法を用いた被験者は，ページオブジェクトパターンというデザインパターンを用いてテストスクリプトの実装を行っている．ページオブジェクトパターンを用いることで，テストケースと各ページの機能を切り離すことができ，テストスクリプトの保守性が向上することが知られている [7]．これらのことから，短期間で繰り返しリリースするような開発プロジェクトにおいては，ページオブジェクトパターンを採用したプログラミングによる手法が適していると言える．しかし，プログラミングによる手法で保守性の高いテストスクリプトを実装するには，依然としてスキルのある開発者が時間をかけて行う必要があり，End-to-End 自動テストを開発現場に導入する際の障壁となっている．この問題を解決するために，クローリングを用いてページオブジェクトを自動生成する研究 [8] が行われているが，クローリングが困難な大規模なアプリケーションに適用しにくい，また利用に人手が必要といった問題がある．

本研究では，人手による End-to-End テスト（以下，手動テスト）のログを用いて，ページオブジェクトパターンを採用したテストスクリプトを自動生成する手法を提案する．現在の産業界におけるテストは人手に頼る部分が多く存在している [9], [10]．その理由として，テスト自動化の導入コストが大きいために導入を躊躇する開発現場が多いこと，画面表示のテスト・ユーザビリティのテスト・手順が決まっていないテスト等の自動的な検証が困難なテストが存在すること等が考えられる．

提案手法は，開発において必要不可欠である手動テストを活用することで，提案手法を適用するための事前準備をほとんど必要とせずにテストスクリプトを生成することができる．これにより，人手によるテストスクリプト実装および既存のページオブジェクト生成手法の問題点を解決している．提案手法では，ある画面でテスターが実際に行った操作をその画面の機能を利用する操作とみなし，ページオブジェクトのメソッドとして生成する．また，テスト対象の画面遷移を解析することで，なるべく少ない本数でテスト対象の機能を網羅するようなテストケース群を生成する．これにより，自動テストにおいて再利用性の高いメソッドおよびテストケースを生成することができる．提案手法を用いることで，開発現場への自動テストの導入障壁を大きく下げることが期待できる．

本研究では，OSS の Web アプリケーションに対して，テスト経験者 4 名に手動テストを行ってもらい，提案手法






```
1 class AddOwnerPage {
2   // 入力
3   get _firstName() { return $('#firstName'); }
4   get _lastName() { return $('#flastName'); }
5   // 中略
6
7   // リンク
8   get _home() { return $('span*=HOME'); }
9   // 中略
10
11  // ボタン
12  get _addOwner() { return $('button*=Add Owner'); }
13
14  // 機能
15  goHome() {
16    this.home.click();
17    return new HomePage();
18  }
19  // 中略
20
21  addOwner(firstName, lastName,
22           address, city, telephone) {
23    this._firstName.setValue(firstName);
24    this._lastName.setValue(lastName);
25    // 中略
26    this._addOwner.click();
27    return new OwnerPage();
28  }
29 }
```

図 1　PetClinic のオーナー追加ページとページオブジェクト

を適用することで有用なテストスクリプトを生成できるかどうかを評価した．その結果，既存のページオブジェクト生成手法よりも多くの有効なページオブジェクトを生成できることを示した．また，ページオブジェクトを利用したテストケースを自動生成し，少ないテストケース数でテスト対象の大部分の機能を確認できることを示した．

2. ページオブジェクト

ページオブジェクトとは，テスト対象の各画面をそれぞれオブジェクト指向プログラミングにおけるオブジェクトとして表現したものを指す．図 1 は PetClinic という OSS

の Web アプリケーションのオーナー追加ページと，そのページオブジェクトの例である．オーナー追加ページは 5 つの入力フォーム，4 つのリンク，1 つのボタンで構成されている．また，オーナー追加ページの持つ機能として，他ページへの遷移およびオーナーの追加が考えられる．ページオブジェクトでは，ページをクラスとして定義し，画面中の操作対象となる要素をアクセサ，画面が提供する機能をメソッドとして定義する．

図 1 では，テストフレームワーク WebdriverIO を用いて JavaScirpt で記述した例を示している．`_firstName` アクセサは，名前入力フォームを表しており，`$('#firstName')` は画面中で HTML の id が「firstName」である要素を捕捉する WebdriverIO の機能である．`#firstName` のように画面中の要素を一意に特定するための情報をロケータといい，id・name・テキスト・XPath 等を用いることができる．定義されたアクセサは，同じページオブジェクト内のメソッドからのみ用いられる．`addOwner` メソッドは，各入力フォームに入力する値を引数として受け取り，各入力フォームにそれらを入力した後，オーナー追加ボタンをクリックするメソッドである．テストケース中でオーナー追加画面で何らかの操作を行う場合は，ここで定義されたメソッドを用いる．また，メソッドの戻り値は一般的に遷移先のページオブジェクトである．これにより，メソッドチェーンによりテストケースを記述することができる．したがって，同じ機能でも入力値によって遷移先が異なる場合は，戻り値だけが異なるメソッドを複数作成する．ページオブジェクトを用いることで，処理の共通化を行いつつ，テストケースと画面の機能を切り離すことができる．これにより，テスト対象の画面や機能に変更があった場合でもページオブジェクト中のロケータやメソッドを修正するだけで良く，テストケースへの変更を最小限にすることができる．

3. 関連研究

ページオブジェクトを自動生成する既存手法として，APOGEN が存在する [8]．APOGEN はテスト対象の Web アプリケーションをクローリングし，画面の構成要素を自動的に抽出することで，ページオブジェクトを自動生成する．この際，APOGEN は類似度に基づく画面のクラスタリングを行い，同じクラスタに属する画面は統合して 1 つのページオブジェクトを生成する．一般的に，機能的に類似した画面が複数ある場合，それらを統合した 1 つのページオブジェクトを作成するべきである．なぜなら，類似した複数のページオブジェクトが存在することはテストスクリプトの保守性を下げることにつながるためである．

APOGEN を用いることでページオブジェクト実装のコストを削減できる可能性があるが，テストケースはユーザが実装する必要がある．また，ページオブジェクト生成においても何点かの問題点を抱えている．

まず，APOGEN を利用するために人手が必要となる問題がある．画面遷移を行うために特定の入力が必要な場合，クローラにあらかじめ操作対象と入力値の組み合わせを教える必要がある．また，APOGEN が提案したクラスタに問題がある場合は人手で修正する必要がある．

次に，APOGEN が生成するページオブジェクトの品質の問題がある．クローリングでテスト対象の画面遷移や機能を網羅できない場合，その画面のページオブジェクトを生成することができない．また，APOGEN は HTML の Form タグで囲まれた要素に対する操作を 1 つの画面の機能としてメソッド化するが，特に動的なページではそれに当てはまらない操作が多く，そのような操作がメソッド化されない．さらに，可能な画面遷移が全てメソッド化されるため，多くのリンクが存在するアプリケーションに対して適用すると実際のテストでは用いないメソッドが多数生成されてしまう．

提案手法は APOGEN とは異なり，ページオブジェクトだけでなくテストケースも自動生成する．また，クローリングを行わずに開発において必要不可欠な手動テストのログを用いるため，事前準備がほぼ不要である．また，テスターの操作を元にすることで，画面の構造にとらわれず正確に画面の機能を抽出することができるため，自動テストにおいて有用なメソッドを生成しやすい．

Yandrapally らは，レコード＆リプレイツールで生成したテストスクリプトの保守性を向上させるためにテストスクリプトの共通化を自動化する手法を提案している [11]．Yandrapally らの手法では，テスト対象の DOM を解析することで，共通化すべき操作を特定している．評価実験により，テストスクリプトの一部をサブルーチン化することで，ステップ数を 49%–75%削減できることを示した．

工数を最小限にする End-to-End テストスクリプトの自動生成手法として，クローリングによる手法が研究されている [12], [13], [14]．これらは，テスト対象アプリケーションを動的に探索することで，テスト対象の機能性を網羅するようなテストスクリプトを自動生成することを目的としている．Lin らは，自然言語処理技術を用いることで，より効率的にテスト対象アプリケーションの機能性を網羅する手法を提案している [15]．しかし，これらの技術で生成されるテストスクリプトは完全なものではなく，開発者がアサーションを追加するといった修正が必要である上，保守性が考慮されていないため，そのまま継続的な開発に組み込むことが困難である．

4. 提案手法

図 2 に提案手法の全体図を示す．提案手法を用いる事前準備として手作業によるテストのログ（以下，テストログ）を取得する必要がある．著者らは，過去の研究でテス

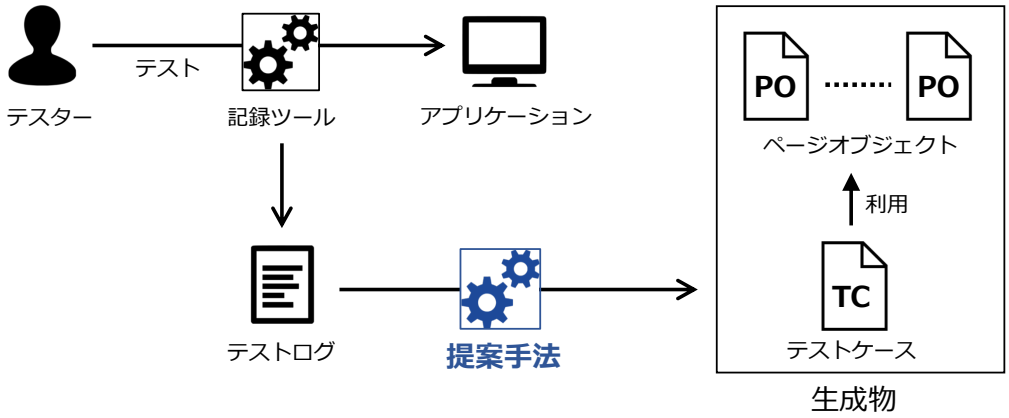


図 2　提案手法の全体図

```
"pageInfo":{
    "title":"PetClinic :: a Spring Framework
demonstration",
    "url":"http://localhost:8080/owners/1/pet
s/new"
},
"operation":{
    "type":"input",
    "input":"2020/3/3",
    "elementInfo":{
        "tagname":"INPUT",
        "text":"",
        "xpath":"/HTML/…/DIV/INPUT",
        "attributes":{
            "class":"",
            "id":"",
            "name":"birthdate",
        }
    }
}
```

図 3　記録されたテストログの例

トログを取得する機能を持つツールを開発している [16]. テストログ取得ツールにより得られる情報は，操作対象要素のタグと属性，テスターによる操作の種類と入力値および操作を行った画面である．テスト中にテストログ取得ツールを起動しておくことで，テスターはツールの存在を特に意識すること無くこれらの情報を取得することができる．図 3 にテストログで記録された 1 つの操作の情報の例 (PetClinic のペット追加ページでペットの誕生日を入力した操作) を示す．

提案手法はテストログを入力として，ページオブジェクトおよびそれを利用するテストケースを生成する．提案手法は，ページオブジェクト生成とテストケース生成の 2 つのフェーズで構成されている．ページオブジェクト生成フェーズでは，操作された要素の情報および操作手順を用いて各画面のページオブジェクトを生成する．テストケース生成フェーズでは，テストログから得られた画面遷移を元に有効なテストケースを抽出し，ページオブジェクト中のメソッドを用いて自動生成する．それぞれのフェーズについて以下で詳細に説明する．

4.1　ページオブジェクト生成フェーズ

提案手法では，テスト中に訪れた全画面のページオブジェクトを生成する．Web アプリケーションの中には，シングルページアプリケーション等，HTTP リクエストにより明示的に画面遷移しないものも存在するため，画面の定義を明らかにする必要がある．本研究では，同一画面の定義として，タイトルの一致または URL の一致をユーザが選択できるようにした．また，正規表現を与えることで，正規表現にマッチするタイトルもしくは URL を全て同一とみなすことも可能である．

ページオブジェクト中のアクセサとして，テスト中一度でも操作された画面中の要素を定義する．アクセサは，WebdriverIO の機能を用いて，指定したロケータが指す要素を返すようにする．耐変更性を考慮し，ロケータは (i)id・(ii)name・(iii) テキスト・(iv) タグ名による絶対 XPath の順に優先して用いる．テキストは，<a> タグおよび，<button> タグのように内部にテキストを保持できる要素が操作されたときのみ用いられ，リンクテキストおよび要素内部のテキストが一致するかどうかで要素を特定する（一意に特定できない場合は XPath を使用する）．

ページオブジェクトパターンにおいて，メソッドはその画面で行われる一連の操作であり，その画面が提供する機能を表す．操作 O は，操作種類 t, 入力値 i，操作対象の要素 e，操作を行った画面 s を用いて，$O = \langle t, i, e, s \rangle$ と表される．ここでは操作種類 t はクリックおよび入力の 2 種類とし，t がクリックの場合は入力値 i は存在しない．また，要素 e は，タグ名 tag，テキスト $text$, 属性 $attr$，および XPath を用いて，$e = \langle tag, text, attr, XPath \rangle$ と表される．提案手法では，テスターがある画面に遷移してから離れるまでに行った一連の操作をその画面の機能を利用するための動作とみなし，これを操作列と呼ぶこととする．全ての操作列をメソッド化した場合，テストログが長くなると大量の重複したメソッドが生成される恐れがある．したがっ

て，ある操作列 A の遷移先画面が他の操作列 B と同じかつ，A が B に包含される場合には A を生成しないようにすることで，重複するメソッドの生成を抑制する．

ここで，「操作列 A が B に包含される」とは，A, B の操作対象の列をそれぞれ A', B' とした時に，A' が B' の部分列になっていることと定義する．例えば，画面中の要素 e_1, e_2, e_3, e_4 があり，$A' = [e_1, e_2, e_4]$，$B' = [e_1, e_2, e_3, e_4]$ としたとき，A' が B' の部分列となるため，操作列 A はメソッド化されない．ここでは操作対象の要素 e のみを考慮し，入力値 i が異なっていたとしても包含されるとみなす．このように操作列をメソッド化することで，遷移先と操作対象を網羅しつつ，役割が重複しにくいメソッドを生成することができる．

生成されるページオブジェクト中の識別子名について説明する．クラスの識別子名は画面の定義として用いたタイトルまたは URL を元に生成する．アクセサの識別子名は，要素の id，name，テキストのいずれかを元に生成する．メソッドの識別子名は，リンクをクリックする操作は「go+遷移先の画面名+通番」，それ以外の操作は「do+最後に呼び出すアクセサ名+通番」というルールで生成する．

また，画面遷移が正しく行われたかを確認するアサーションを自動生成する．現在開いている画面と想定されている画面が一致しているかを検証する仕組みを各ページオブジェクトのコンストラクタに導入することで，テストケースからページオブジェクトが利用されたときに画面遷移の正しさを自動的に検証することができる．これにより，想定される画面遷移が行われない，もしくは操作対象の要素が見つからない場合はテスト実行を失敗させることができる．画面中の文字列や要素の存在等を検証に用いたい場合はユーザがアサーションを追加する必要がある．

4.2 テストケース生成フェーズ

生成されたページオブジェクトで定義されたメソッドを組み合わせることでテストケースを生成する．提案手法では，開発者が修正することを前提として，テスト対象の主要なユースケースを含んだ汎用性の高いテストケースを生成することを目指す．具体的には以下の条件を満たすような最小のテストケース群を生成する．

- テストケース群が画面遷移を網羅する
- 1 つのテストケースで同じ画面に二度訪れた場合，それ以降の画面遷移を行わない
- 他のテストケースで確認された画面遷移はなるべく行わない

テストケースを生成するための手順について説明する．まず，テストログを解析し，テストで訪れた範囲でテスト対象の画面遷移図を生成する．画面遷移図に対して深さ優先探索を行うことで，上記の条件を満たすようなパスのリストを取得する．ここで，パスとはテストケースの元となる連続した画面遷移であり，全てのパスは手動テストの開始時の画面を最初の画面とする．各パスについて，それぞれの画面遷移を行うようにページオブジェクト中のメソッドを繋げて呼び出すようにしたものを 1 つのテストケースとする．例えば，画面 P, Q, R があるとし，パス $P \to Q \to R$ をテストケースとする場合，まず画面 P のページオブジェクトに定義された画面 Q に遷移するメソッド m を呼び出す．メソッド m の戻り値は Q のページオブジェクトなので，メソッド m に繋げて Q のページオブジェクトに定義された画面 R に遷移するメソッドを呼び出す．このようにして，各パスをテストケースに変換することで条件を満たすテストケース群を生成する．

同じ画面遷移を行うメソッドが複数ある場合，最初に生成されたメソッドを用いる．メソッドに与える引数は，テストログを解析し，その画面遷移を行ったときにテスターによって実際に最初に入力された値を与える．

表 1 PetClinic の画面および機能

画面	機能
トップ画面	なし
オーナー検索画面	名字でオーナーを検索する
オーナー検索結果画面	オーナー検索でヒットしたオーナー一覧を表示する
オーナー追加・編集画面	オーナーの情報を入力しオーナーを追加・更新する
オーナー情報画面	オーナーに対しペットの追加・編集訪問情報の追加等を行える
ペット追加・編集画面	ペットの情報を入力し，ペットを追加・更新する
訪問情報追加画面	ペットの訪問情報を入力し，訪問情報を追加する
獣医師一覧画面	なし
エラーサンプル画面	なし

表 2 被験者によるテストの概要

被験者	テスト手法	テストケース数	操作数
A	スクリプトテスト	9	135
B	スクリプトテスト	20	258
C	探索的テスト	–	378
D	探索的テスト	–	505

5. 評価実験

提案手法が有用なテストスクリプトを生成できることを示すために評価実験を行った．ここでのテストスクリプトとはページオブジェクトとそれを用いるテストケースを合わせたものを指す．実験の準備として，OSS の Web アプリケーションである Spring PetClinic を対象として 4 名の被験者が手作業によるテストを行った．

PetClinic の全画面および機能を表 1 に示す．どの画面からもヘッダのリンクをクリックすることでトップ画面・オーナー検索画面・獣医師一覧画面・エラーサンプル画面に

遷移することができる．ここで，オーナー追加画面とオーナー編集画面はボタンのラベルがそれぞれ「Add Owner」と「Update Owner」である以外に画面の構造が一致するため，合わせて 1 つのオーナー追加・編集画面とみなす．また，ペット追加画面とペット編集画面も同じ理由で 1 つのペット追加・編集画面とみなす．これらの画面は機能と構造が類似しているため，ページオブジェクトを作成する場合も 1 つのページオブジェクトにすべきであると考える．

本実験ではどのようなテストログを用いても提案手法が有効であることを示すため，性質が異なる 2 種類のテスト手法を用いた．手作業によるテスト手法は，スクリプトテストと探索的テストの 2 つに大別できる．スクリプトテストは，事前にどのようなテストを実施するべきかを考えてテスト設計を行い，それに沿ってテスト実行を行う手法である．テスト設計はテスト設計書としてドキュメント化され，必要に応じてテスト設計を具体的な確認手順として記述したテスト手順書が作成される．一方，探索的テストは事前にテスト設計を行わず，テスターがテストを実行しながら臨機応変にテスト設計を行う手法である．本実験では，被験者 4 名のうち 2 名がスクリプトテスト，残りの 2 名が探索的テストでのテストを行った．被験者の 4 名は 3 年以上のテスト経験があり，特に探索的テストを行う 2 名は探索的テストの経験がある．

被験者の 4 名にはあらかじめ対象アプリケーションを操作し，PetClinic の仕様を把握してもらった．テストの前提として，単体テストはサーバサイド・クライアントサイド共に十分行われているとし，被験者にはユーザ目線での機能性やユーザビリティに関するテストを行うように指示した．スクリプトテストを行う 2 名は，把握した仕様を元にテスト設計を行い，テストケースを事前に作成した後，それに沿ってテストを実施した．探索的テストを行う 2 名は，30 分を上限とし，各自の知見を活かしてバグを見つけるように自由にテストを行った．テスト中の操作は 4 節で述べたテストログ取得ツールで全て記録した．被験者 A–D から得たテストログをそれぞれテストログ A–D とする．4 名が行ったテストの概要を表 2 に示す．表 2 において，探索的テストでは事前にテスト設計を行わないため，テストケースが存在しない．また，操作数は被験者が画面中の要素を操作することで発生した click イベントおよび change イベントの回数を示している．テストログ A–D に対して提案手法を適用することで，4 パターンのテストスクリプトを自動生成した．提案手法が生成したテストスクリプト群はインターネット上に公開している[*1]．

テストログ B から生成されたテストケースの一例とそれが利用しているメソッドを図 4 に示す．このテストケースでは，オーナー検索画面に遷移，名字でオーナーを検索，ペット追加・編集画面に遷移，名前と誕生日を入力してペットを追加する手順がメソッドチェーンで記述されている．また，テストケース 1 行目の `new _()` はトップページのページオブジェクトを生成する命令である．

[*1] https://github.com/knukio/ses2020_testscripts

表 3 提案手法および APOGEN によって生成されたページオブジェクト中のメソッドの分類

生成元	完全	冗長コードを含む	要修正	不要	リンクをクリック
APOGEN	6	0	7	0	18
テストログ A	7	2	3	0	7
テストログ B	12	0	7	1	12
テストログ C	9	2	4	6	16
テストログ D	10	2	7	3	12

5.1 ページオブジェクト生成

提案手法が生成したページオブジェクトが有効であるかの評価を行うために，既存のページオブジェクト生成ツール APOGEN との比較を行った．まず，提案手法を適用することでそれぞれのテストログから 9 画面分のページオブジェクトを得た．この際，各オーナーのオーナー情報画面，オーナー追加・編集画面，ペット追加・編集画面はこれらがそれぞれ 1 つの画面とみなされるように URL の正規表現を用いて画面の定義を行った．また，APOGEN を PetClinic に対して適用しページオブジェクトを生成した．この際，APOGEN のクローラに情報を与えてなるべく多くの画面に到達できるようにし，クラスタリングによって表 1 に示す画面に分類できるようにクラスタを定義した．しかし，エラーサンプル画面およびオーナー検索結果画面は APOGEN の仕様上の限界により生成することができなかった．これは文献 [8] の評価結果と一致している．したがって，到達できなかった 2 画面を除く 7 画面についてページオブジェクトを生成した．

テストログ A–D および APOGEN から生成されたページオブジェクト中のメソッドを以下の基準で分類した．

完全 引数・命令・戻り値について修正すべき部分がないメソッド．

冗長コードを含む 動作は正しいが，動作に影響を与えない無意味な操作を含んでいるメソッド．

要修正 利用するために引数・命令・戻り値のいずれかを修正する必要があるメソッド．

不要 同じ機能を確認するメソッドが複数ある場合の 2 つ目以降のメソッド．

リンクをクリック ヘッダ中のリンクをクリックし画面遷移を行うメソッド．

リンクをクリックに分類されたメソッドは全て完全なメソッドであるが，数が多い一方で実際のテストケースで利用される可能性が低いと考えられるため，その他のメソッドと区別することとした．

テストケース

```
new _()
  .go_owners_lastName_find()
  .doFind_Owner2({
    lastName: 'TEST2',
  })
  .go_owners_d_pets()
  .doAdd_Pet({
    name: 'SAMPLE',
    birthDate: '2020-05-10',
  });
```

ページオブジェクト中のメソッド

```
go_owners_lastName_find() {
  this.FIND_OWNERS.click();
  return new _owners_lastName_find();
}
```

```
go_owners_d_pets() {
  this.Add_New_Pet.click();
  return new _owners_d_pets();
}
```

```
doFind_Owner2({lastName}) {
  this.lastName.setValue(lastName);
  this.Find_Owner.click();
  return new _owners_d();
}
```

```
doAdd_Pet({name, birthDate}) {
  this.name.setValue(name);
  this.birthDate.setValue(birthDate);
  this.Add_Pet.click();
  return new _owners_d();
}
```

図 4 生成されたテストケースの一例と利用しているメソッド

分類結果を表 3 に示す．表 3 の結果より，提案手法は APOGEN と比較してより多くの完全なメソッドを生成できることが分かった．提案手法では APOGEN がクローリングで到達できなかった画面に対しても到達するようなメソッドを生成できた．

無意味な命令を含むメソッドは，提案手法を用いた場合にのみ生成された．今回の実験では，同じフォームに同じ値を複数回入力している場合がほとんどであった．これは，被験者がテスト中に一度入力した入力値を修正するといった操作を行い，それが一連の操作とみなされてしまったために起こっていた．

要修正のメソッドは，APOGEN と提案手法の両方で生成されているが，提案手法の方が数が少ない傾向が見られた．APOGEN が画面遷移を正しく認識できていないためか 7 件の要修正のメソッド全てにおいて適切な戻り値が無かった．提案手法が生成した要修正のメソッドは，引数の数が不十分で一部の入力フォームに対して値が入力できない例が多かった．これは，テスト時に特定の画面遷移を行う際，入力フォームの全てに入力を行わなかったことが原因と考えられる．例えば，オーナーの登録を失敗するときの動作を確認するテストとして，名前を空欄にしてオーナー登録を行うテストのみを行った場合，手法の性質上，生成されるメソッドは名前を入力する操作を含まない．しかし，住所を与えない，不正な文字を与えるなど，登録を失敗する入力は他にも考えられる．したがって，オーナーの登録を失敗する操作を行うメソッドにおいても，全ての入力フォームに対して引数で入力を与えられるようにし，名前を空欄にするテストは引数で空文字列を与えることで実現すべきであると考える．

不要なメソッドは提案手法のみで生成された．今回の実験では，オーナー検索結果画面において，各オーナーをクリックするメソッドがそれぞれ別のメソッドとして生成された例がほとんどであった．しかし，仮に APOGEN がオーナー検索結果画面に到達できていた場合，全てのユーザーに対してそれぞれをクリックするメソッドを生成し，提案手法より多くの不要なメソッドを生成していたと考えられる．

ヘッダ中のリンクをクリックするメソッドは，ページオブジェクトが 7 つしか無いにもか関わらず APOGEN が最も多く生成していた．テストではこれらのリンクが有効であることを確認できれば良いため，これに分類されるほとんどのメソッドは利用されないと考えられる．

これらの結果より，提案手法を用いることでテスト手法やテスターに左右されず有効なページオブジェクトを APOGEN よりも多く生成できることが示された．

表 4 テストケースの分類

生成元	完全	データ依存	要メソッド修正	合計
テストログ A	3	3	0	6
テストログ B	5	1	1	7
テストログ C	7	1	1	9
テストログ D	7	1	1	9

5.2 テストケース生成

次に，提案手法が生成したテストケースが有効であるかの評価を行う．どのようなテストケースを自動化すべきであるかはプロジェクトに依存するが，本実験では少ない本数でテスト対象の各機能を正しく利用した場合に想定される操作を行うテストケースが作成できたかを評価した．なぜなら，このようなテストケースは汎用的でどのようなプロジェクトでも有用であると考えられるためである．また，これらのテストケースを流用することで，機能を正しく利用しない場合（オーナー登録に失敗するケースなど）のテストケースも容易に実装できる．

まず，テストログ A–D から生成されたテストケースを以下の基準で分類した．

完全 メソッドの呼び出し手順・引数の値・データベースの状態を修正せずに実行可能なテストケース

データ依存 データベースの状態を初期状態から変える，もしくはメソッドの引数で与える値を変えることで，画面遷移を変えること無く実行可能なテストケース

要メソッド変更 呼び出すメソッドの一部を，画面遷移を変えないような別のメソッドに置き換えることで実行可能なテストケース．

テストケースの分類結果を表 4 に示す．表 4 より，テストケースの数は 6 本から 9 本に収まっており，手作業によるテストの操作数がテストケースの本数にあまり影響しな

表 5　生成されたテストケースによって確認できる機能

画面	番号	機能	A	B	C	D
オーナー検索画面	1	オーナー検索で 1 件ヒットした場合，オーナー情報画面が表示される	✓	✓	✓	✓
	2	オーナー検索で 2 件以上ヒットした場合，オーナー検索結果画面で表示する	–	✓	–	–
	3	オーナー検索で何も入力しなかった場合，オーナー検索結果画面で全件表示する	✓	×	✓	✓
	4	オーナーが追加できる	✓	✓	×	✓
オーナー検索結果画面	5	オーナー名をクリックしてオーナー情報画面に遷移できる	–	✓	✓	✓
オーナー情報画面	6	ペットが追加できる	✓	✓	✓	×
	7	ペットが編集できる	–	×	×	✓
	8	オーナー情報が編集できる	✓	✓	✓	✓
	9	訪問情報が追加できる	✓	✓	✓	✓
オーナー追加・編集画面	10	値を入力してオーナーが追加できる	✓	✓	✓	✓
	11	値を入力してオーナーが編集できる	✓	×	✓	✓
ペット追加・編集画面	12	値を入力してペットが追加できる	✓	✓	✓	×
	13	値を入力してペット情報が編集できる	–	×	×	✓
訪問情報追加画面	14	値を入力して訪問情報を追加する	✓	✓	✓	✓
ヘッダ	15	トップ画面・オーナー検索画面・獣医師一覧画面・エラーサンプル画面に遷移	✓	✓	✓	✓

いことが分かる．

一部のテストケースがデータ依存に分類された原因として，手作業でのテスト中に追加したオーナーに対してペット追加等のテストを行っていることが挙げられる．提案手法はデータの依存関係を考慮しないため，データベースの初期状態で存在しないオーナーに対してペット追加等を行うテストケースを生成する例があった．これは，テストケースで与える入力値を初期状態で存在するオーナーに置き換える，またはデータベースの初期状態を変更するといった対処を行うことで実行可能になる．

また，一部のテストケースが要メソッド変更に分類された原因として，異なる機能を持つ画面を 1 つの画面として定義している場合がある点が挙げられる．例えば，本実験ではペットの追加と編集は別の機能として定義しているが，ペット追加・編集画面は 1 つの画面と定義している．ペット追加・編集画面において，情報を入力した後「Add Pet」ボタンを押してオーナー画面に遷移するメソッドと「Update Pet」ボタンを押してオーナー画面に遷移するメソッドは別のメソッドとして定義されるが，画面遷移が同じであるためテストケース生成時には区別されない．これにより，ペットを追加すべき場面でペットを更新するメソッドを呼び出すテストケースが生成されてしまう例があった．この場合，ペットを更新するメソッドを呼び出す代わりにペットを追加するメソッドを呼び出すように変更することで実行可能になる．

表 5 はテストログ A–D から生成されたテストケース群によって各機能が確認できたかを示している．PetClinic の機能は，被験者 A と B がスクリプトテスト実施時に作成したテストケースから洗い出した．表中の「✓」は生成されたテストケース群で機能を確認できること，「×」は生成されたテストケース群で機能を確認できないこと，「–」は被験者によるテストでその機能が確認されておらず，その機能を確認するテストケースが手法の性質上生成できないことを表している．例えば，トップ画面からオーナー検索画面に遷移して検索を行い，検索でヒットした 1 人のオーナーに対してペットを追加するテストケースが生成されていれば番号 1，6，12 の機能が確認できているとみなす．また，表 5 の結果は，表 4 で **データ依存**および**要メソッド変更**に分類されたテストケースを正しく修正した場合の結果を示している．表 5 の結果より，少ないテストケース数で大部分の機能が確認できていることが分かる．

一部の機能について，手作業によるテストでは確認されているにも関わらず提案手法が生成したテストケースでは確認できない場合があった．これらはほとんどの場合，提案手法が画面遷移を網羅することを基準にテストを生成していることに起因していた．例えば，あるテストケースで，オーナー情報ページから「Add New Pet」ボタンでペット追加・編集画面に遷移した場合はその画面遷移は確認済みになる．一方，オーナー情報ページから「Edit Pet」ボタンを押した場合もペット追加・編集画面に遷移するが，この画面遷移は既に確認済みのため，ペットの編集機能を確認するテストケースは生成されない．これにより表 5 の番号 2 と 3，6 と 7，12 と 13 の機能は手法の性質上，どちらか一方しか確認できない．しかし，生成されたテストケースをわずかに変更することでもう一方の機能を確認するテストも容易に作成することができると考える．

これらの結果より，提案手法が少ないテストケースでテスト対象の機能を確認するようなテストケースを自動生成できることが分かった．生成されたほとんどのテストケースは少しの修正を行うことで実行可能であり，生成されなかったテストケースに関しても生成されたテストケースおよびページオブジェクトを利用することで容易に作成することができる．

6. 結論

本研究では，テスターが手作業で行ったテストのログを用いてページオブジェクトパターンを採用したテストスクリプトを生成する手法を提案した．評価実験により，既存手法の問題を解決し，より多くの有効なページオブジェクトおよびメソッドを生成できることを示した．また，生成したページオブジェクトを利用するテストケースを生成し，少ないテストケース数でテスト対象の機能の大部分を確認できることを示した．生成されたテストケースやページオブジェクトは再利用しやすい形式になっており，これらを利用することで新たなテストケースを追加することも容易である．提案手法を用いることで，ソフトウェア開発で必要不可欠な手作業によるテストを行うだけで有用なテストスクリプトを生成することができ，テストスクリプトの実装・保守のコストを下げることが期待できる．

今後の課題として，テストログに含まれるテスターの知見を有効活用することで，メソッド化する一連の操作をより正確にする，同じ画面遷移を行うメソッドを適切に使い分ける，といった改善を行うことで，より完全なページオブジェクトおよびテストケースを生成することを目指す．

参考文献

[1] Yang, B., Hu, H. and Jia, L.: A Study of Uncertainty in Software Cost and Its Impact on Optimal Software Release Time, *IEEE Transactions on Software Engineering*, Vol. 34, No. 6, pp. 813–825 (2008).

[2] Bertolino, A.: Software Testing Research: Achievements, Challenges, Dreams, *Future of Software Engineering*, pp. 85–103 (2007).

[3] Orso, A. and Rothermel, G.: Software Testing: A Research Travelogue (2000–2014), *Proceedings of the on Future of Software Engineering*, Hyderabad, India, ACM, pp. 117–132 (2014).

[4] Selenium: http://www.seleniumhq.org/.

[5] Christophe, L., Stevens, R., Roover, C. D. and Meuter, W. D.: Prevalence and Maintenance of Automated Functional Tests for Web Applications, *IEEE International Conference on Software Maintenance and Evolution*, pp. 141–150 (2014).

[6] Leotta, M., Clerissi, D., Ricca, F. and Tonella, P.: Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution, *20th Working Conference on Reverse Engineering*, pp. 272–281 (2013).

[7] Leotta, M., Clerissi, D., Ricca, F. and Spadaro, C.: Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study, *IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 108–113 (2013).

[8] Stocco, A., Leotta, M., Ricca, F. and Tonella, P.: Clustering-Aided Page Object Generation for Web Testing, *Web Engineering*, Springer, Cham, pp. 132–151 (2016).

[9] Berner, S., Weber, R. and Keller, R. K.: Observations and Lessons Learned from Automated Testing, *Proceedings. 27th International Conference on Software Engineering*, pp. 571–579 (2005).

[10] Andersson, C. and Runeson, P.: Verification and Validation in Industry - a Qualitative Survey on the State of Practice, *Proceedings International Symposium on Empirical Software Engineering*, pp. 37–47 (2002).

[11] Yandrapally, R., Sridhara, G. and Sinha, S.: Automated Modularization of GUI Test Cases, *Proceedings of the 37th International Conference on Software Engineering*, Piscataway, NJ, USA, IEEE Press, pp. 44–54 (2015).

[12] Iyama, M., Kirinuki, H., Tanno, H. and Kurabayashi, T.: Automatically Generating Test Scripts for GUI Testing, *IEEE International Conference on Software Testing, Verification and Validation Workshops*, pp. 146–150 (2018).

[13] Milani Fard, A., Mirzaaghaei, M. and Mesbah, A.: Leveraging Existing Tests in Automated Test Generation for Web Applications, *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, Vasteras, Sweden, Association for Computing Machinery, pp. 67–78 (2014).

[14] Dallmeier, V., Pohl, B., Burger, M., Mirold, M. and Zeller, A.: WebMate: Web Application Test Generation in the Real World, *IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pp. 413–418 (2014).

[15] Lin, J.-W., Wang, F. and Chu, P.: Using Semantic Similarity in Crawling-Based Web Application Testing, *IEEE International Conference on Software Testing, Verification and Validation*, pp. 138–148 (2017).

[16] 切貫弘之，倉林利行，丹野治門，熊川一平，永田啓悟：探索的テストと操作の記録を組み合わせた新たなテスト手法の提案，信学技報, vol. 119, no. 56, KBSE2019-7，pp. 43–48 (2019).