Vol.2016-SE-194 No.10
2016/11/18

Webアプリケーションに対する，画像比較に基づいた回帰テスト結果自動合否判定

堀 旭宏[1,a)]　高田 眞吾[1,b)]　倉林 利行[2,c)]　丹野 治門[2,d)]

概要：アプリケーションに対する回帰テストの研究は数多く存在するが，それらの多くはテストケース選択や優先順位づけ，およびテスト自動実行に関する研究であり，テスト結果自動合否判定に関する研究は少ない．一般的に，回帰テスト結果の合否判定はアプリケーション変更前後の実行結果を比較することで行われることが多い．比較により，実行結果が一致していれば合格，そうでなければ不合格と判定する．近年，JavaScript や CSS を用いた動的な Web アプリケーションが増加しているが，そのような Web アプリケーションに対して自動で回帰テスト結果の合否判定を行う手法は存在しない．そこで本研究では，動的に変化する Web アプリケーションに対しても自動で回帰テスト結果の合否判定を行う手法を提案する．提案手法の中心は，テスト中の Web アプリケーションの GUI 状態が変化するたびにスクリーンショットを取得し，各状態でアプリケーション変更前後の画像比較を行うことである．評価の結果，提案手法は人間よりも高い精度で合否判定を行うことができ，また，実用的な速さで合否判定を行うことができることが分かった．

キーワード：Web アプリケーションテスト，回帰テスト，JavaScript，CSS

1. はじめに

現在，多くの Web アプリケーションが開発されている．アジャイル [1] やリーン開発 [2] などの短期繰り返しリリース手法の普及により，Web アプリケーションは頻繁に変更されるようになり，回帰テストの需要が高まっている．回帰テストとは，プログラムに変更を加えた際、変更が加えられる前にパスしたテストと同様のテストを，変更後プログラムがパスできるかを検証すること [3] であり，プログラム変更前に実施したテストを変更後に再度実施することによって行われる．回帰テストはプログラムが変更されるたびに行う必要があるため，膨大なコストがかかる．そこで，回帰テストの自動化が求められる．

また，近年，JavaScript や CSS を用いた動的な Web アプリケーションが増加している．「動的な Web アプリケーション」とは，マウスやキーボードの操作に応じて，同一ページ内で GUI の表示内容が変わる Web アプリケーションのことである．例えば，会員登録画面で新規パスワードの入力に応じてパスワードの安全度が即座に表示されたり，スライドショーのように時間経過に応じて写真が自動的に変わったりするものがある．このような動的な Web アプリケーションに対し，開発者は GUI の観点からテストを行う必要がある．

回帰テストの研究は数多く存在するが，それらの多くはテストケース選択や優先順位づけ，およびテスト自動実行に関する研究であり，テスト結果自動合否判定に関する研究は少ない．特に，動的なアプリケーションに対して GUI の観点でテスト結果自動合否判定を行う研究は存在しない．そこで，本研究では動的に変化する Web アプリケーションに対しても自動で回帰テスト結果の合否判定を行う手法を提案する．

一般的に，回帰テスト結果の合否判定はプログラム変更前と変更後の実行結果を比較することで行われることが多い．比較により，実行結果が一致していれば合格，そうでなければ不合格と判定する．Web アプリケーションを対象とした場合の比較方法には主に 2 つある．1 つ目は，DOM 要素のプロパティ情報を用いた比較である．2 つ目はスクリーンショットを利用した画像比較である．本研究では GUI の観点から合否判定を行うために，後者の画像比較方法を用いる．

画像比較によって Web アプリケーションの回帰テスト

[1] 慶應義塾大学
[2] NTT
[a)] horicun@doi.ics.keio.ac.jp
[b)] michigan@ics.keio.ac.jp
[c)] kurabayashi.toshiyuki@lab.ntt.co.jp
[d)] tanno.haruto@lab.ntt.co.jp

結果を自動合否判定するツールはいくつか存在するが，いずれも動的な Web アプリケーションに対して正確に合否判定を行うことはできない．なぜなら，既存ツールは GUI の状態変化を考慮しておらず，一連の入力が行われた最後にしか比較を行っていないからである．

そこで本研究では，テスト中の Web アプリケーションの GUI の状態が変化するたびにスクリーンショットを取得し，各状態でプログラム変更前後の画像比較を行う手法を提案する．ここで，「状態」とは Web アプリケーションに対してマウスやキーボードによるイベントが与えられた直後の表示内容を指す．

この手法により，クリックやオンマウスによって表示が変化するページや，入力に応じて即時フィードバックを返すページなどに対して正しく回帰テスト結果の合否判定を行うことができる．しかし，この手法では正しく比較を行えない要素も存在する．それは，時間経過によって表示内容が変わる要素（動く要素）である．例えば，写真が次々に表示されるスライドショーや，動画が埋め込まれたページに対しては，比較するタイミングが少しでもずれてしまうと正しく比較を行うことができない．

そこで本研究では，このような動く要素に対しては，「変更前に動いていた要素が変更後にも動いているか」および「変更前に動いていなかった要素が変更後にも動いていないか」のチェックのみを行い，それが満たされていれば合格，満たされていなければ不合格とする．この手法では変更前に動いていた要素が，変更後も 正しく 動いているかをチェックすることができない．しかし，ある程度の品質を担保しつつ，本手法の適用可能領域を拡大することができる．

また，画像比較の方法として，本研究では，変更前・変更後の実行結果のスクリーンショットを複数の領域に分割し，領域ごとに位置ずれを補正したうえで完全比較を行う．完全比較とは，すべてのピクセルが一致していれば一致，そうでなければ不一致と判定するものである．この比較方法により，従来より高い精度で比較を行うことができる．

本研究の貢献は以下の通りである．

(1) Web アプリケーションの一連の操作を複数の状態に分割し，状態ごとに比較を行うことで，動的に変化する Web アプリケーションに対して合否判定を行うことができる．
(2) 時間経過によって表示内容が変わる要素（動く要素）に対しては「変更前に動いていた要素が変更後にも動いているか」を合否判定の基準に用いることで，1 の適用可能領域を拡大することができる．
(3) 画像比較時に位置ずれを補正したうえで完全比較を行うことで，比較精度を向上させる．

2. 関連研究

テストオラクルとは，テストの予想出力 [4]，すなわち，テストにおける合否判定の基準となるものである．一般的に，テストオラクルの生成は困難であると言われているが [5]，回帰テストオラクルの場合，プログラム変更前のテスト実行結果を再利用することができる．すなわち，プログラム変更後のテスト実行結果がプログラム変更前のテスト実行結果と一致していれば合格，そうでなければ不合格と判断することができる．ただし，これはプログラム変更前のテストがすべて合格であることが前提である．本研究では，この手法を用いて合否判定を自動で行う．

動的な Web アプリケーションに対してオラクルを生成する研究はいくつか存在する [6]，[7]，[8]．しかし，これらの研究はすべて DOM 要素のプロパティ情報に基づいて比較を行っており，動的な Web アプリケーションに対して画像ベースでオラクルを生成する研究は存在しない．これは，動的な Web アプリケーションを画像に基づいて比較することが困難であるためであると考えられる．しかし，受け入れ段階のテストを想定した場合，最終的に GUI が正しく表示されているかは見た目が正しいかで判断されるべきである．見た目が正しいかを確かめる方法としては，プロパティ情報よりも画像に基づいた比較方法のほうがより直接的でふさわしいと考えられる．そこで，本研究では画像に基づいて比較を行う．

Choudhary らは Web ブラウザのクロスブラウザ問題を解決する手法を提案した [9]．彼らの研究の目的は本研究の目的とは異なるが，スクリーンショットを各領域に分割し，領域ごとに比較を行う点は共通する．ただし，彼らは画像比較方法として輝度のヒストグラムに基づいて画像間の距離を算出し，それが閾値以下であれば一致とする手法をとっていたが，これでは，本来異なる画像に対しても一致していると判定してしまう可能性がある．例えば，画像中の文字の位置がずれているバグに対しては，輝度のヒストグラム自体は変わらないため一致と判定してしまう．そこで，本研究では，画像間のずれを補正したうえで 2 枚の画像が全く同じかを調べ，同じであれば一致していると判定する．これにより，バグの見逃しを防ぐことができる．

「2 枚の画像が全く同じか」を一致の基準にすることは，場合によってはやや厳しすぎることもある．例えば，数ピクセル違っていても，その違いが人間には検知できないのであれば「一致」であると判定するべきという考え方もある．これは回帰テストオラクルをどのように定義するかによる問題である．しかし，数ピクセルであれ何らかの違いが存在しているということは，変更後のファイルに文法ミスが紛れ込んでいる可能性がある．特に，HTML や CSS では文法ミスがあったとしてもエラーメッセージが表示さ

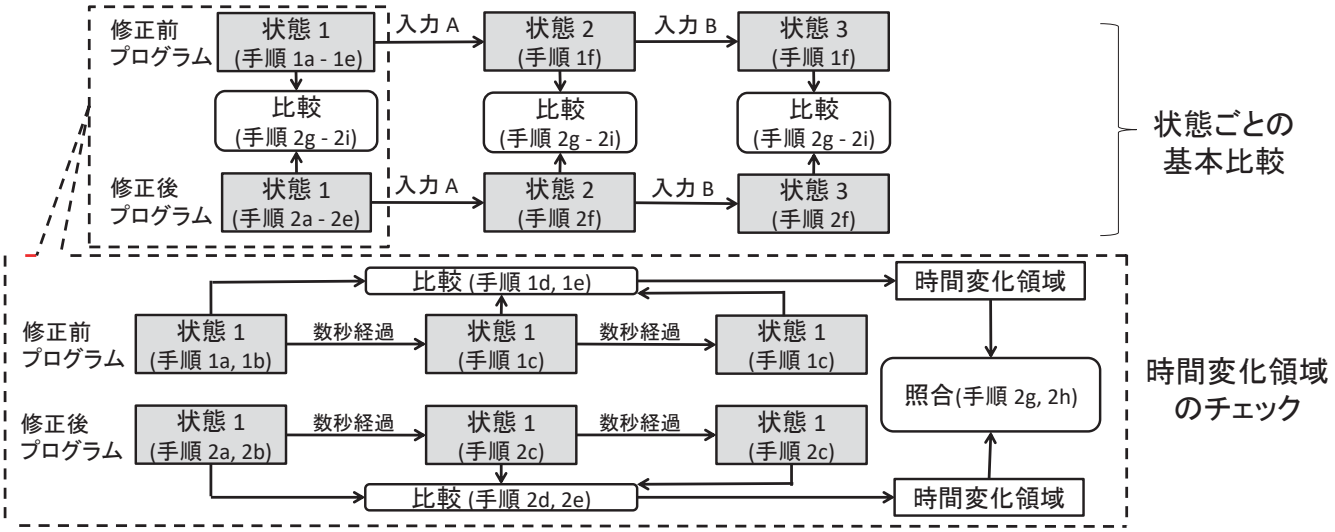

図 1 提案手法の概要

れないため，ミスを見逃しがちである．そこで，本研究では「2 枚の画像が全く同じか」を一致の基準に用いることで，より厳しいテストオラクルを設定し，バグの見逃しを防ぐことを目指す．

我々の以前の研究 [10] では，Web アプリケーションの回帰テストを自動判定する手法を提案したが，動的に変化する Web アプリケーションに対して正しく合否判定することができなかったため，本研究ではその解決を図る．また，以前の研究では比較の際に Choudhary ら [9] と同じ輝度のヒストグラムを用いて比較を行っていたが，先に述べた理由から本研究では画像間の完全一致を判定の基準とする．

Web アプリケーションに対し，画像比較をもとに回帰テストを自動実行し，自動で合否判定を行うツールはいくつか存在する [11]，[12]，[13]．しかし，いずれも動的に変化する Web アプリケーションに対して正確に合否判定を行うことはできない．そこで本研究では動的に変化するアプリケーションに対しても画像比較をもとに回帰テストの自動合否判定を行う手法を提案する．

3. 提案手法

3.1 概要

Web アプリケーションの一連の操作を複数の状態に分割し，状態ごとに比較を行い，一致していれば合格，そうでなければ不合格と判定する．提案手法の処理の流れを図 1 に示す．基本的な手順は以下の通り．

(1) 各テストケースに対し，以下のサブステップを変更前のプログラムに対して実行する．
  (a) Web アプリケーションの最初の Web ページを開く．
  (b) テストケース中の最初の入力を与え，スクリーンショットおよび DOM ツリーを取得する．
  (c) 数秒間放置したうえで，再びスクリーンショットを取得する．これを複数回行う．
  (d) 手順 1b と手順 1c で取得したスクリーンショットをそれぞれ複数の領域に分割する．(3.3 節)
  (e) 手順 1b と手順 1c で取得したスクリーンショットを対応する領域ごとに画像比較し (3.4 節)，一致しなかった領域を「時間変化領域」と認定する．本研究では，時間経過に応じて表示が変わる領域を「時間変化領域」と定義する．
  (f) テストケースのすべての入力に対して手順 1b〜手順 1e を行う．
(2) 変更後のプログラムに対しても手順 1a〜手順 1f と同じ操作を行う．(それらを手順 2a〜手順 2f とする．) さらに，それらに加え以下のサブステップを追加する．
  (g) 「変更前の時間変化領域が変更後も時間変化領域であるか」を確かめる．
  (h) 「変更前に時間変化領域でなかった領域が変更後も時間変化領域でないか」を確かめる．
  (i) 変更前・変更後ともに時間変化領域でない領域に関しては，変更前と変更後で状態ごとに画像比較し (3.4 節)，表示内容が同じであるかを確かめる．
(3) 手順 2g〜手順 2i において，すべての状態・領域で合格したテストケースについては合格と判定する．そうでない場合は不合格と判定する．

3.2 例

図 2 に示すようなホテル予約アプリケーションを例に，3.1 節で示した提案手法の処理の流れを説明する．このアプリケーションではプルダウンボタンをクリックするとホテル名のリストが表示され，リスト中のホテル名をクリックするとそのホテル名が自動的に入力され，リストの表示

が消える．

また，カレンダーボタンをクリックするとカレンダーが表示され，カレンダー中の日付をクリックするとその日付が自動的に入力される．今回の例ではアプリケーションのリファクタリングを行った際にバグが混入し，「チェックイン」のカレンダーの日付をクリックしてもその日付が入力されなくなってしまったと仮定する．アプリケーションが変更されたので，回帰テストを行う．回帰テストで用いるテストケースの一部を表 1 に示す．1 つのテストケースは複数の入力からなる．また，今回は紙面の都合上 1 つのテストケースしか示していないが，実際には複数のテストケースが存在する．

まず，変更前プログラムのテスト実行を行う．初めに，初期状態（図 2（a））のスクリーンショットおよび DOM ツリーを取得する．次に，数秒経つのを待ち，再びスクリーンショットおよび DOM ツリーを取得する．そして，取得したスクリーンショットを複数の領域に分割し，それぞれの領域ごとに画像比較を行う．今回の例では時間経過によって表示内容が変わる領域は存在しないので，時間変化領域は検出されない．次にテストケース 1 の最初の入力が行われる．すなわち，「ホテル名選択」のプルダウンボタンがクリックされる．この状態でスクリーンショットと DOM ツリーを取得し，数秒待ってから再び取得する．そしてスクリーンショットを複数の領域に分割し，それぞれの領域ごとに画像比較を行う．この操作を 1 つ 1 つの入力ごとに行う．すべてのテストケースのすべての入力に対して操作を終えたところで変更前プログラムのテスト実行が終了する．

次に，変更後プログラムに対しても変更前プログラムと同様の操作を行う．まず，初期状態のスクリーンショットおよび DOM ツリーを取得し，数秒待ってから再び取得し，領域に分割し，領域ごとに画像比較を行う．ここでも時間変化領域は検出されない．これにより，「変更前に時間変化領域でなかった領域が変更後も時間変化領域でないか」が確かめられたことになる．また，すべての領域が変更前・変更後ともに時間変化領域ではなかったため，各領域に対して変更前後での比較を行う．比較の結果，すべての領域で「一致」と判定されれば，表示内容全体が同じであるという判定をする．この操作を 1 つ 1 つの入力ごとに行う．すると，「チェックイン」のカレンダーの 12 月 1 日をクリックした後の状態での比較で，変更前後での比較が「不一致」となる領域が検出される．従って，テストケース 1 の合否結果は「不合格」と判定される．

3.3 領域分割

提案手法の手順 1d，手順 2d ではスクリーンショットを複数の領域に分割している．この方法として，我々の以前の研究 [10] で採用した手法を用いる．これは DOM ツリーの情報を用いて各領域の面積が閾値より大きくなるように分割する手法である．これにより，スクリーンショットをある程度の大きさを持った領域に分割することができる．

図 2 ホテル予約アプリケーションの例

表 1 テストケースの一部

テストケース #1	「ホテル名選択」のプルダウンボタンをクリック
	プルダウンメニューから「The Hotel Prime」を選択
	「チェックイン」のカレンダーボタンをクリック
	カレンダーの 12 月 1 日をクリック
	「チェックアウト」のカレンダーボタンをクリック
	カレンダーの 12 月 3 日をクリック

3.4 画像比較

提案手法の手順 1e，2e，2i で行う画像比較は，図 3 に示す流れで行われる．図 3 の 1，2，3 は位置ずれの補正の手順を表しており，4，5 は完全比較の手順を表している．

3.4.1 位置ずれの補正

本研究では位置ずれの補正を行う．これは，各領域をスクリーンショットから切り出す際に，数ピクセルのずれが発生してしまうことを許容するためである．まず，画像 2 の 5 ピクセル内側の領域を抽出し，テンプレートとして設定する．次に，Open CV [16] のライブラリにある cvMatchTemplate メソッドを用いて，画像 1 の中で最もテンプレートにマッチする場所を探し，マッチした場所で画像 1 と画像 2 を重ね合わせる．そして，重ね合わせた時にはみだした領域を削除する．以上の操作により，画像 1 と画像 2 の大きさをそろえることができる．

3.4.2 完全比較

本研究では完全比較を行なう．画像間の差分をとり，差分が 0 であれば一致していると判断し，そうでなければ不一致であると判断する．

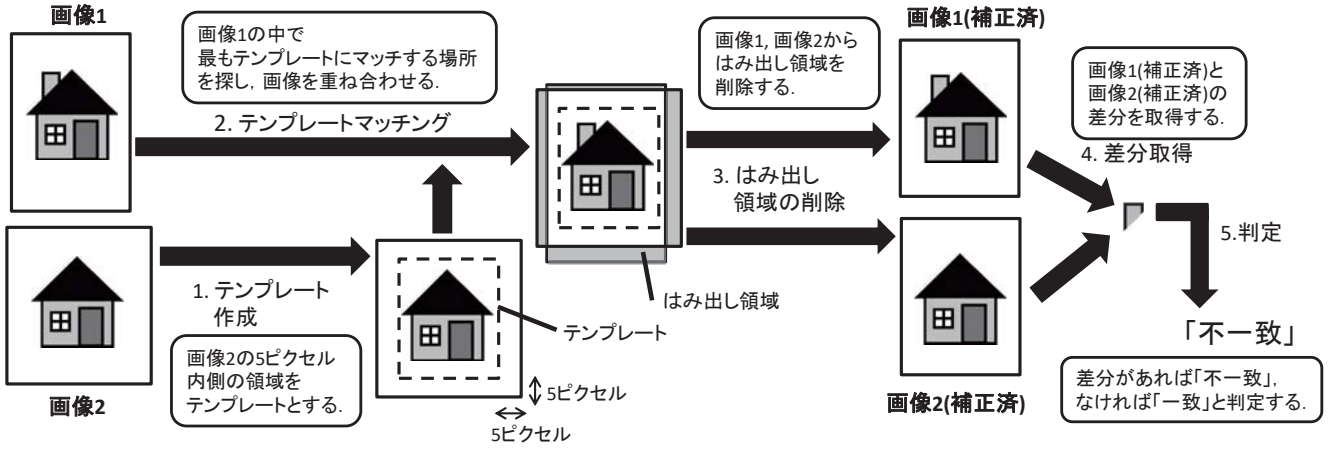

図 3 画像比較の方法

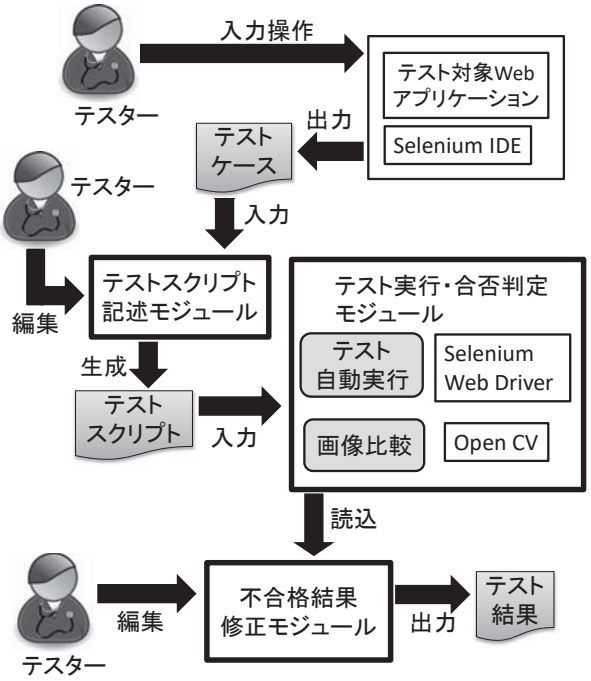

図 4 ツールの処理の流れ

3.4.3 高速化

図 3 では，位置ずれの補正を行った後に完全比較を行うと示したが，実際には高速化のために先に完全比較を行い，この時点で一致したものに関しては位置ずれの補正を行うことなく「一致」と判定する．そして，不一致であったものに関しては位置ずれの補正を行ったうえで再び完全比較を行い，一致か不一致かを判定する．

4. 実装

本研究では，提案手法の実用性を確かめるために，提案手法をツールに実装した．ツールの処理の流れを図 4 に示す．本研究で実装した箇所は「テストスクリプト記録モジュール」「テスト実行・合否判定モジュール」「不合格結果修正モジュール」の 3 か所であり，提案手法本体は「テスト実行・合否判定モジュール」に実装されている．

4.1 テストケースの作成

まず，テスターは Selenium IDE [14] を用いてテストケースの作成を行う．Selenium IDE は Web アプリケーションに対する入力を記録することができるツールである．テスターは Selenium IDE を起動した状態でテスト対象 Web アプリケーションに対して入力を行う．すると，その入力がテストケースとしてファイルに出力される．

4.2 テストスクリプト記述モジュール

次にテスターはテストスクリプト記述モジュールを用いてテストケースを編集し，テストスクリプトを生成する．このモジュールでは 3.1 節で示した手順 1c の「数秒間」の秒数および「複数回」の回数を設定することができる．また，変更前プログラム，変更後プログラムの URL を設定することもできる．また，各入力の直後が「状態」として自動的に設定される．編集が終了したら編集内容がテストスクリプトとしてファイルに出力される．

4.3 テスト実行・合否判定モジュール

次に，テスト実行・合否判定モジュールがテストスクリプトをもとにテストを自動実行し，合否判定を行う．処理の流れは 3.1 節で示した手順の通りである．テスト自動実行には Selenium Web Driver [15] を利用し，画像比較には Open CV [16] を用いる．合否判定結果はデータベースに保存される．

4.4 不合格結果修正モジュール

最後に，テスターは不合格結果修正モジュールを用いて合否判定結果を編集する．提案手法では開発者が意図的に変更した部分に対しても「不合格」と判定してしまう．本来，開発者が意図的に変更した部分は回帰テストの対象外であるが，本提案手法ではその区別を手動で行う．テス

表 2　評価対象アプリケーション

機能		アプリケーション
機能 1	オンマウス	[18][19][20][21]
機能 2	スライドショー	[22][23][24][25]
機能 3	動画埋め込み	[26][27][28][29]
機能 4	即時フィードバック	[30][31][32][33]

ターはこのモジュールを用いることで，このような不合格判定を手動で合格に修正することができる．編集が終了したら最終的なテスト結果をファイルに出力する．

5. 評価実験

評価実験では，判定精度および判定に要する時間を評価項目とした．また，比較対象として，人間が手動で判定を行った場合の判定精度も調査した．

5.1 対象アプリケーション

我々は，動的に変化するページが持つ代表的な 4 つの機能に着目した．各機能に対し，4 つの Web アプリケーションを評価対象アプリケーションとして用意した．機能及び評価対象アプリケーションを表 2 に示す．

このうち，[18]， [22]， [26]， [30] を予備評価用アプリケーションとして使用し，残りを本評価用に用いた．

5.2 評価方法

以下の手順で評価を行った．

(1) 変更後アプリケーションの作成
(2) テストケースの作成
(3) 閾値の設定（予備実験）
(4) ツールの判定精度・判定時間の測定（本実験）
(5) 人間が判定を行った場合の精度の測定（比較対象）

5.2.1 変更後 Web アプリケーションの作成

計 16 個のアプリケーションそれぞれを変更前のアプリケーションとし，それぞれにバグを埋め込み，変更後のアプリケーションを作成した．バグの埋め込み方法として，機械的にプログラムの一部を書き換える方法を用いた．書き換え規則には「>」を「<」に置き換えるなどの単純な置き換え規則や，ある一行を削除するという削除規則を用いた．1 つの変更前アプリケーションにつき，バグ入りプログラムは作成可能なだけすべて作成した．このうち，バグが実際に表示上に現れるものだけを残し，それ以外は全て削除した．さらに，バグの現れ方が全く同じものも削除した．最終的に残ったプログラムを変更後アプリケーションとした．その数を表 3 の「変更」に示す．

5.2.2 テストケースの作成

Web アプリケーションの GUI に焦点を当てたテストケース自動作成手法は存在しないため，手動で状態遷移図を作成し，それに基づいてテストケースを作成した．これには，スニークパステスト [17] という方法を用いた．生成されたテストケース数を表 3 の「TC」に示す．

5.2.3 実験手順

予備実験および本実験は以下の手順で行った．まず，変更前アプリケーションに対してテストケースを実行した．次に，各変更後アプリケーションに対し，ツールを用いてテストケースをすべて実行し，合否判定を行った．そして，それらの結果を手動で念入りに確かめ，正しく判定を行っているかを調べた．

5.2.4 予備実験

16 個の Web アプリケーションのうち，[18]， [22]， [26]， [30] を予備実験用アプリケーションとして使用した．予備実験は 3.3 節の領域分割時に用いる閾値を設定するために行った．閾値を 200,000 に設定したとき，判定精度が 100%を示した．

5.2.5 本実験

予備実験により決定した閾値を用いて，残りの 12 個の Web アプリケーションに対して本実験を行い，判定精度および判定に要した時間を測定した．

5.2.6 人間による判定

ツールの比較対象として，人間が判定を行った場合の精度を測定した．9 人の被験者に 3 種類のアプリケーションについて手動でテスト実行およびテスト合否判定を行ってもらった．3 種類のアプリケーションのうち，1 つ目は練習用に使用し，残りの 2 つを実験用に用いた，そして，PC の画面を 2 分割し，左に変更前，右に変更後の Web アプリケーションを表示させ，テストケースを入力してもらい，挙動が同じかどうかを確かめ，同じであれば○，そうでなければ×を記入してもらった．ただし，被験者の負担を減らすため，テストケース数は上限を 20 個，変更後アプリケーション数は上限を 5 個とした．

5.3 評価結果

5.3.1 判定精度

ツールおよび人間が判定を行ったときの判定精度を表 3 に示す．表 3 において，P は合格（Pass），F は不合格（Fail）を表しており，P/F とは，「正しい結果が合格であり，かつ，ツール（または人間）が不合格と判定したテストケースの数」を示す．正解率，適合率，再現率，F 値は次のように計算される．

- 正解率 $= \big(P/P + F/F\big) \div$ テストケース数
- 適合率 $= F/F \div \big(F/F + P/F\big)$
- 再現率 $= F/F \div \big(F/F + F/P\big)$
- F 値 $= \big(2 \times$ 適合率 $\times$ 再現率 $\big) \div \big($ 適合率 $+$ 再現率 $\big)$

全体でみると，ツールの正解率は 96.2%であったのに対し，人間の正解率は 86.8%であり，ツールの方が高い正解率を示した．適合率ではツールは 93.1%であり，人間は 97.8%であったため，人間の方がよかったが，再現率ではツールは 99.9%であり，人間の 79.5%を大きく上回った．

表 3 ツールおよび人間の回帰テスト結果判定精度

	アプリ		セットアップ			結果（テストケース数）				判定精度			
			TC	変更	合計	P/P	F/F	P/F	F/P	正解率	適合率	再現率	F 値
ツール	機能 1	[19]	9	5	45	22	23	0	0	1.000	1.000	1.000	1.000
		[20]	36	10	360	121	198	41	0	0.886	0.829	1.000	0.906
		[21]	21	21	441	120	321	0	0	1.000	1.000	1.000	1.000
	機能 2	[23]	16	5	80	43	25	2	0	0.975	0.946	1.000	0.972
		[24]	9	5	45	4	33	8	0	0.822	0.805	1.000	0.892
		[25]	9	5	45	16	24	5	0	0.889	0.828	1.000	0.906
	機能 3	[27]	4	5	20	5	14	0	1	0.950	1.000	0.933	0.966
		[28]	4	5	20	3	15	2	0	0.900	0.882	1.000	0.938
		[29]	2	3	6	1	5	0	0	1.000	1.000	1.000	1.000
	機能 4	[31]	12	14	168	100	68	0	0	1.000	1.000	1.000	1.000
		[32]	10	6	60	42	18	0	0	1.000	1.000	1.000	1.000
		[33]	11	25	275	246	29	0	0	1.000	1.000	1.000	1.000
	合計		143	109	1565	723	783	58	1	0.962	0.931	0.999	0.964
人間	合計		126	57	842	336	395	9	102	0.868	0.978	0.795	0.877

5.3.2 時間

ツールは 1 テストケースあたり 26.3 秒でテスト実行・結果判定を行った．これは実用的な速さであるといえる．

5.4 考察

ツールが正しく判定できなかった原因を次に挙げる．

- Selenium Web Driver による操作がうまく機能しなかった．（40 テストケース）
- 領域画像の切り出しがうまくいかなかった（10 テストケース）
- マッチングがうまくいかなかった（1 テストケース）

人間が正しく判定できなかった原因としては，注意不足による見落としが考えられる．人間の集中力には限界があり，多くのテストケースを継続して判定すると，誤った判定を下してしまうことが考えられる．特に，動的に変化するアプリケーションの判定では，操作しながら判定を行わなければならず，集中力が散漫になりやすいと考えられる．そのため，ツールによる自動判定は有用であるといえる．

5.5 妥当性の脅威

最初の妥当性の脅威は Web アプリケーションの数である．本研究では 16 アプリケーションを用意し，12 アプリケーションに対して本評価を行った．結果の一般性を向上させるために，さらに多くのアプリケーションに対して評価を行う必要がある．

2 個目の妥当性の脅威は機能の数である．本研究では動的に変化するアプリケーションが持つ 4 個の機能に焦点を当てて評価を行った．しかし，他にも機能は存在する．例えば，SNS（ソーシャルネットワーキングサービス）の組み込み機能などが挙げられる．そこで，評価結果の一般性を向上させるために，さらに多くの機能に対して評価するべきである．

3 個目の妥当性の脅威は操作可能な入力イベントの種類である．現在ツールが扱う入力はマウスやキーボードによる操作のみであるが，評価結果の一般性の向上のために，さらに多くの入力イベント操作を実装し，本提案手法が適用できるかを確認するべきである．さらなる入力イベントとして，タッチパネルによる入力や音声入力などが挙げられる．

4 個目の妥当性の脅威はテストケースの作成の仕方である．Web アプリケーションに対して GUI の観点でテストケースを自動で作成する方法が存在しなかったため，本研究では手動で状態遷移図を作成し，それをもとに手動でテストケースを作成した．しかし，結果の一般性の向上のために，テストケースの作成を他の人に行ってもらうか，別の手法でテストケースを作成するべきである．

5 個目の妥当性の脅威はバグの埋め込み方法である．本研究ではあらかじめ定めた変更規則に基づいて機械的にプログラムの一部を書き換える方法を用いたが，この方法では単純なバグばかりを埋め込んでしまう．そこで，より複雑なバグに対しても本提案手法が有効であるかを確かめる必要がある．

6 個目の妥当性の脅威は被験者に関してである．本研究では学生を被験者として採用したが，プロのテスターのほうが被験者としてふさわしい．ただし，被験者に与えたタスクは特別な知識を必要としないシンプルなものであったため，これについてはさほど大きな脅威であるとは考えられない．また，本研究では 9 人の被験者に手動で判定してもらったが，より多くの被験者に判定してもらうべきである．

6. 結論

本研究では動的に変化する Web アプリケーションに対して状態ごとに比較を行う事で回帰テストの合否判定を行

う手法を提案した．ただし，時間変化領域に対しては「変更前の時間変化領域が変更後も時間変化領域であるか」を合否基準とした．また，領域ごとに比較を行う際に，ずれを補正してから比較を行うことで比較精度を向上させた．評価実験では提案手法の実用性が確かめられた．

今後の課題として，以下が挙げられる．

テストケース選択

プログラムの意図的な変更の影響を受けたテストケースは，次回の回帰テスト時にあらかじめ取り除いておくべきである．そのような機能を追加することでより効率よく回帰テストを行うことができる．

状態間の遷移

本研究の提案手法では状態間の遷移が正しく行われたかを判定することができない．例えば，スライドショーで写真が別の写真に遷移するときのエフェクトにバグが含まれていた場合，本研究の提案手法ではそのバグを検知することができない．

時間変化領域

時間変化領域については変更後も動いているかの確認しかできず，正しく表示されているかの確認ができない．しかし，動画埋め込みについては，埋め込み URL を解析することで変更前と同じ動画が埋め込まれているかを確認することができる．そこで，どのような遷移をしているかについても確認する手法が求められる．

広告の除去

我々の以前の研究では広告などのランダムに表示が変わる要素を合否判定の際に除外する手法を提案したが，その手法は本研究にも適用可能であり，適用することでより精度の高い合否判定を行うことができる．

参考文献

[1] L. Williams and A. Cockburn, “Agile software development: it’s about feedback and change,” Computer, vol. 36, no. 6, pp. 3943, 2003.
[2] J. P. Womack, D. T. Jones, and D. Roos, The Machine That Changed the World: HarperPerennial, 1990.
[3] Eric J. Braude, Software Engineering. An Object-Oriented Perspective: John Wiley & Sons Inc, 2000.
[4] Boris Beizer, Software Testing Techniques: Van Nostrand Reinhold, 1990.
[5] S. R. Shahamiri, W. M. N. W. Kadir, and S. Z. Mohd-Hashim, “A comparative study on automated software test oracle methods,” in Proc. Fourth International Conference on Software Engineering Advances (ICSEA 2009), 2009, pp. 140145.
[6] S. Mirshokraie and A. Mesbah, “JSART: JavaScript assertion-based regression testing,” in Proc. 12th International Conference on Web Engineering (ICWE 2012), 2012, pp. 238252.
[7] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, “JSEFT: Automated Javascript unit test generation,” in Proc. IEEE 8th International Conference on Software Testing, Verification and Validation (ICST 2015), 2015, pp. 110.
[8] D. Roest, A. Mesbah, and A. V. Deursen, “Regression testing Ajax applications: Coping with dynamism,” in Proc. Third International Conference on Software Testing, Verification and Validation (ICST 2010), 2010, pp. 127136.
[9] S. Choudhary, H. Versee, and A. Orso, “A cross-browser web application testing tool,” in Proc. 26th IEEE International Conference on Software Maintenance (ICSM 2010), 2010, pp. 16.
[10] A. Hori, S. Takada, H. Tanno, and M. Oinuma, “An oracle based on image comparison for regression testing of web applications,” in Proc. 27th International Conference on Software Engineering and Knowledge Engineering (SEKE 2015), 2015, pp. 639645.
[11] “Applitools,” https://applitools.com/.
[12] “PhantomCSS,” https://github.com/Huddle/PhantomCSS.
[13] “Screenster,” http://www.creamtec.com/products/screenster/.
[14] Selenium IDE. https://addons.mozilla.org/ja/refox/addon/selenium-ide/.
[15] Selenium. http://www.seleniumhq.org/.
[16] Open CV. http://opencv.org/.
[17] Robert Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools (Addison-Wesley Object Technology): Addison Wesley Pub Co Inc, 1999.
[18] “Winter Olympics Template,” http://images.all-free-download.com/free-website-templates-preview/winter olympics template 1990/.
[19] “wood working,” http://www.opendesigns.org/od/wp-content/designs/2/2539/.
[20] “CSS3 photo,” http://www.opendesigns.org/od/wp-content/designs/2/2493/.
[21] “Music Life,” https://w3layouts.com/music-life-entertainment-categoryflat-bootstrap-responsive-web-template/.
[22] “metamorph simplegrey,” http://www.opendesigns.org/od/wp-content/designs/2/2422/.
[23] “Snail on the rain,” http://www.opendesigns.org/od/wp-content/designs/2/2877/.
[24] “Dilasag,” http://www.openwebdesign.org/design/6339/dilasag-simple/.
[25] “The New Reporter,” https://w3layouts.com/the-new-reporter-aentertainment-category-flat-bootstrap-responsive-web-template/.
[26] “Videostube,” https://w3layouts.com/preview/?l=/videos-like-youtube-mobile-website-template/.
[27] “My Play a Video,” https://w3layouts.com/videos-like-youtube-mobilewebsite-template/.
[28] “Youtuber,” http://demo.themewagon.com/preview/youtuber-youtube-marketer-responsive-html5-template.
[29] “Life Coach,” http://demo.themewagon.com/preview/life-coach-a-coaching-website-templates.
[30] “Remember The Milk,” http://jqueryvalidation.org/files/demo/milk/.
[31] “Valida,” http://awin.com.br/valida/sample/.
[32] “jQuery-DataEntry,” http://ugrose.com/content/demos/jqdataentry/.
[33] “jQuery Validation Plugin,” http://jqueryvalidation.org/files/demo/.