Vol.2016-SE-194 No.9
2016/11/18

画像処理を活用したUIレイアウト崩れ検出支援手法の提案

丹野 治門[1,a)]

概要：スマホ・タブレット・PCや，OS，ブラウザ等の多数のバリエーションがあるさまざまな環境上で動作するアプリケーション (Webアプリケーション，Androidアプリケーション等) のテストでは，それぞれの環境でアプリケーションの画面がUIレイアウト崩れを起こさず正しく表示されることを確認する必要がある．本論文では，画像処理技術を活用し，正しく表示されている画面の画像と，テスト対象の画面の画像を入力として，画面要素の消失や位置ずれといった差異を自動検出することでUIレイアウト崩れの発見を支援する手法を提案する．UIレイアウト崩れを人工的に埋め込んだアプリケーション画面を題材として，完全な目視確認に対して，提案手法を使って確認した場合にどれだけ誤り発見率が上がり，確認に要する時間が削減されるかについての評価実験も行った．

キーワード：テスト結果確認，UIレイアウト，多環境，他端末，画像処理

1. はじめに

スマホ・タブレット・PCや，OS，ブラウザ等の多数のバリエーションがあるさまざまな環境上で動作するアプリケーション (Webアプリケーション，Androidアプリケーション) のテストでは，それぞれの環境でアプリケーションの画面が正しく表示されることを確認する必要がある．アプリケーションの画面が正しく表示されているかどうかの確認には，アプリケーションのロジックが正しく動作し，正しい計算結果が表示されているかどうかの確認と，画面の構成部品のレイアウトが正しく描画されて表示されているかどうかの確認がある．前者はテスト自動実施ツール (Selenium，Appium 等) で自動実行するスクリプトにアサーションを記述することで確認を機械的に行うことも可能であるが，後者は最終的に端末のスクリーンに描画された画面を目視で確認する必要があるため，確認の労力が大きい．また，このような画面のUIレイアウト崩れをテストで見逃してリリースしてしまった場合，「決定ボタンが画面の外にはみ出してしまい押せない」，「商品の金額を示す文字列の上に他の画面要素が重なり一部隠れてしまった」など，場合によっては単に見栄えが悪くなるだけではなく，アプリケーションの価値が大きく損なわれる危険性がある．近年，環境のバリエーションの数は増え続けており，OS，ブラウザなどの環境の組み合わせも考慮すると多

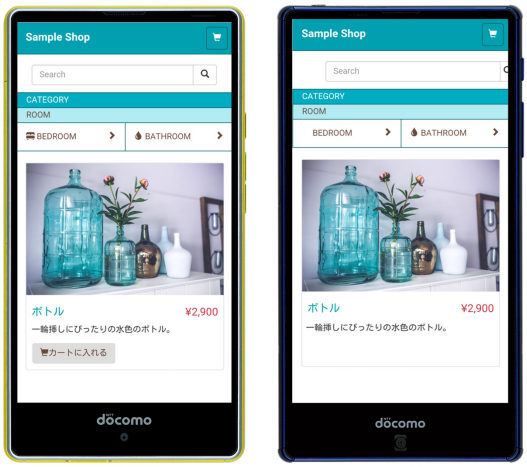

図 1 正解画面と比較対象画面の差異

くのテストを実施し，その結果の画面においてUIレイアウト崩れがないことを確認する必要があり，これはテスト担当者にとって大きな負担となっている．

例えば，オンラインショッピングシステムの商品選択画面を，2つのスマートフォン端末A，Bでそれぞれ動作させ，その結果画面を確認するテストを考えてみる．端末A,Bにおける商品選択画面の表示結果を図1に示す．端末Aがリファレンス端末（正しい画面）であり，端末Bが比較対象の画面である．この画面では実際には，図2に示すように (1) テキストボックスのずれ，(2) アイコンの消失，(3) ボタンの消失，という3箇所のUIレイアウト崩れがある．実際のアプリケーションの1つの画面は，図3に示す

[1] NTT 研究所
NTT Laboratories, Minato, Tokyo 108–0075, Japan
[a)] tanno.haruto@lab.ntt.co.jp

ⓒ 2016 Information Processing Society of Japan

図 2 正解画面と比較対象画面の差異

図 3 確認すべきテスト画面の分量

ように縦に長く全て確認するにはスクロールを要することも多い．このような画面同士の確認を，「アプリケーションの画面の数」×「端末の数」の分だけ目視で全て確認することは，テスト担当者の負担が大きく，確認漏れが発生してしまう可能性もある．

既存ツールとして，2 つの画像を画素単位で比較しその差分を表示するもの (例えば DiffImage [1]) がある．しかし，このような方法だと，画面内の画面要素の変動が大きい場合に，その影響で他の画面要素が全てずれるなどし，ほとんどの箇所が差異として検出されてしまうことがある．また，様々な環境において表示された画面同士を比較する場合は，画面の解像度差や，文字のフォント差異，画面要素の微妙な大きさの差異など，UI レイアウト崩れと無関係のものも全て差異として検出されてしまうため，本当に確認したい UI レイアウト崩れであるような差異を検出することが困難になるという問題がある．

本研究ではこのような問題を解決するため，単純な画素単位の画像比較ではなく，画面要素の単位で正解画像と比較対象画像の差異のある個所を全て自動検出し，テスト担当者が UI レイアウト崩れを検出することを支援する手法を提案する．これにより，これまでテスト担当者が完全にに目視で確認を行っていたときの稼動を削減し，UI レイアウト崩れの検出漏れを減らすことを目指す．本論文では，NTTR&D フォーラム 2016 に出展した技術 [3,4] について，手法の詳細を詳しく述べ，評価実験を行っている．

本研究の貢献は以下のとおりである．

(1) 提案手法している手法は，通常，テストで実施証跡として記録される，アプリケーション画面のスクリーンショット画像のみを入力としており他の入力資材は不要であるため，アプリケーションの実装技術に依存せずにテストで活用することが可能である．また，提案手法では，正解画像と比較対象画像の差異をテキストボックス，ボタンなどといった画面要素の単位で抽出し，画面要素単位で差異を抽出し，それらの差異に追加・消失・移動・拡縮等の差異理由も付与して出力する．そのため，テスト担当者は効率よく，かつ漏れなく，UI レイアウトの崩れの可能性がある箇所を確認していくことができる．

(2) 提案手法では画像処理の技術 [8] を活用し，最初に解像度差を考慮して正解画面の画像と比較対象画面の画像のサイズを合わせた後，それぞれの画面において矩形領域を抽出し，次に 2 つの画面で特徴量抽出とそのマッチングを行い，特徴量を用いて正解画面と比較対象画面における矩形領域を対応付けることで，画面要素ごとの領域を対応づけて，差異を画面要素単位で抽出するという工夫をしている．また，対応付けられた矩形のペアについては場合は移動量，拡縮率等の差異の理由となっている情報も計算しておくことで，これらの情報も含めて，2 つの画面の差異を画面要素の単位で出力する．

(3) 提案手法をツールとして実装した．そして，完全に目視でテスト結果画面を確認する場合に対して，ツールの差異抽出結果を見ながらテスト結果画面を確認したとき，どれだけ確認稼動及び UI レイアウト崩れの検出漏れを減らせる効果があるかを実験で確かめた．

以下，2 節で既存手法と本研究の位置付けを述べる．3 節で提案手法について説明し，4 節で評価実験について述べる．そして最後に 5 節で結論を述べる．

2. 既存手法と本研究の位置づけ

テスト結果確認を自動判定する，もしくは支援する手法は多く研究されてきた [6]．本節では，提案手法に関連する技術として，多環境においてアプリケーションを動作させたときの UI レイアウト崩れの検出や，回帰テストにおけるデグレード検出などを目的に，アプリケーションの実行結果画面を対象としてテスト結果確認として画面上の誤

ⓒ 2016 Information Processing Society of Japan

り箇所を特定することを支援する手法について紹介する．

2.1 アプリケーション実装技術に依存した手法

Web アプリケーションを対象とするものなど，アプリケーションの実装技術に依存した手法について紹介する．

回帰テストを目的としたものは，Hori らの手法 [9] があり，これは Web アプリケーションを対象としており，DOM ツリーの情報を使って画面内の画面要素を特定し，その画面要素ごとに比較を行って，画面要素単位で差異を抽出し，デグレードを検出することが可能である．

WEBDIFF [7], X-PERT [12] はクロスブラウザにおける UI レイアウト崩れの検出を目的として，2 つの画面の DOM ツリーの情報及びアプリケーション画面の画像情報を比較することで，誤り箇所を見つける．また，Alameer ら [5] はロケールが変わったときの UI レイアウト崩れ検出を目的として，正解画面と比較対象画面のそれぞれについて，DOM ツリー上の要素同士の位置関係をグラフ化し，両方のグラフを比較し，グラフ内で要素同士の関係が「交差」に変更されたものをレイアウト崩れとみなして検出している．正解画像を用いずに UI レイアウト崩れ検出を行う手法として，FightingLayoutBugs [2] がある．FightingLayoutBugs は HTML，CSS を入力として，表示エラーや要素の重なりを検出することができる．また，Walsh ら [13] は，レスポンシブルデザインのページにおける UI レイアウト崩れを検出するため，DOM ツリーから Responsive Layout Graph (RLG) というグラフ構造を作成し，そのグラフの差異を活用して誤りを検出している．

これらの手法は回帰テスト，UI レイアウト崩れ検出において有用であるが，Web アプリケーションにおける DOM ツリーなどの画面構造に関する情報を活用しているため，特定の実装技術に依存した手法になっており，Web アプリケーション，Android，iOS など画面表示の実装技術が異なるものごとに，新たな実装を用意する必要がある．

2.2 画面画像のみを用いたアプリケーション実装技術に依存しない手法

特定の実装技術に依存せずに使えるものとして，アプリケーションの画像のみから画面上の誤り箇所を特定しようとしている既存手法を紹介する．

DiffImage [1] は画素単位での差異検出を行っている．また，Mahajan らの手法 [11] も画素単位の差異検出を画像で行っている．これらの手法では，同じ環境（同じ解像度）で，アプリケーション画面で画面要素の追加や削除など，比較する画面間でほぼ変更がない場合の回帰テスト，多環境間の UI レイアウト崩れ検出に有効である．しかしながら，環境の差異によって解像度が変化したり，その解像度変化に伴い画面要素が多少ずれたりするなど画面に変更があった場合や，画面要素の消失があった場合には，多くの箇所が差異として検出されてしまう問題がある．Mahajan ら [10] の手法は，多少の画素単位の差異を吸収しているが，やはり，大きくずれてしまった場合には多くの差異が検出されてしまう．池之上らの手法 [14] では，回帰テストを主な対象として，正解画面と比較対象画面の画面要素矩形を抽出し，両方の画面の特徴量マッチングを行うことで矩形を対応付けることで，矩形単位で 2 つの画面の差異を抽出することが可能であり，ニュース一覧や広告など画面内で動的に大きさが変化する領域があっても，変化のあった画面要素の単位で差異を抽出することができる．

本研究は，解像度差などもある様々な環境でアプリケーションを動作させたときの UI レイアウト崩れの検出を，テスト担当者がなるべく効率よく行えるようにすることを目指している．様々な環境間の UI レイアウト崩れの検出を効率よく行うためには 2 つの画面の差異を画面要素の単位で抽出することに加え，以下が重要となる．

- 抽出した差異を誤りをすべきかどうかは最終的にはテスト担当者の判断となる．そのため，抽出した画面要素にどのように「どのように違うか」の情報が付与されており，人が見たときにそれが誤りとすべきものなのかを判定しやすくする．
- どのような差異を UI レイアウト崩れとみなすかは，開発現場ごとに基準が異なる．そのため，多くの余計な差異を見なくて済むよう無視してもよいような差異は，あらかじめ無視してユーザへ掲示しないようにする．
- 異なるスマートフォン端末上でアプリケーションを動作させ，画面の解像度が異なっている場合でも解像度の違いに起因する無視しても問題ない差異は無視できるようにしたい．例えば，解像度が異なる場合には，画面内の個別の画面要素の拡縮率が異なる場合もある．このような場合には，これらを拡縮率のみ違う画面要素として対応付け，必要に応じて拡縮率が微小のものは無視してユーザに掲示しないようにしたい．

3. 提案手法

本研究では UI レイアウト崩れをテスト担当者が効率よく発見していくことを支援する手法を提案する．提案手法では，図 4 のようにテスト自動実行ツール等で取得したアプリケーションの正解画面と複数の比較対象画面のスクリーンショット画像を入力として，それぞれの比較対象画面について，画面要素単位で正解画面と比較対象画面の差異を，どのような差異かの情報とともにテスト担当者へ掲示する．そして，テスト担当者が掲示された差異に対して OK(問題なし) もしくは NG(UI レイアウト崩れである) を判定し，それらを開発者向けのバグレポートとしてとりまとめるまでを支援する．提案手法には以下の特徴がある．

- 画素単位ではなく画面要素矩形の単位で差異を抽出す

ⓒ 2016 Information Processing Society of Japan

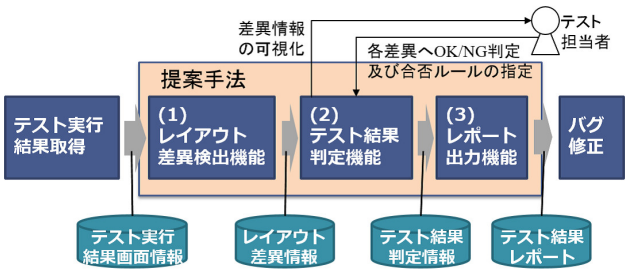

図 4 提案手法の全体像

る．抽出した画面要素単位の差異には移動，拡縮，追加，削除のような「どのように違うか」の情報が付与されており，テスト担当者がその差異を UI レイアウト崩れとすべきかどうかの判定しやすくしている．

- 上述した各差異ごとの「どのように違うのか」の情報を用い，合否判定ルールを設定することで「移動が xx ピクセルのものは許容する」といった設定が可能である．これにより，無視してもよいような差異は，あらかじめ無視して OK/NG を判定するテスト担当者へ掲示しないようにする．
- 異なるスマートフォン端末上のアプリケーション画面の比較を行う際，比較する 2 つの画面の解像度が異なるときにも，全体のサイズと，個別の画面要素のサイズの違いを考慮した上で比較を行って差異を抽出する．

以降, 本節では提案手法の詳細について, 図 4 の (1),(2),(3) の順に説明していく．

3.1 (1) レイアウト差異検出機能

図 4(1) レイアウト差異検出機能では，正解画面の画像 (ここでは，正解画像と呼ぶ) と比較対象画面の画像 (ここでは，比較対象画像と呼ぶ) を入力とし，画像処理の技術を活用し，以下のステップで 2 つの画面の差異を理由付きで検出する．

- Step1：正解画面と比較対象画面のサイズを調整．
- Step2：正解画面において画面要素単位で矩形抽出．
- Step3：正解画面と比較対象画面において特徴量抽出とマッチング．
- Step4：マッチング結果を用い正解画面における矩形領域に対応する比較対象画面内の矩形領域を特定し矩形ペアを作成．
- Step5：矩形ペアで一致/不一致を判定し，不一致の場合には差異理由のタグも付与．

以降，それぞれの Step について簡単に説明していく．

Step1 では，正解画面と比較対象画面のサイズ (縦の画素数と横の画素数) が異なった場合，正解画面の横サイズと比較対象画面の横サイズが一致するよう，比較対象画面を，アスペクト比を固定したままリサイズする．これにより，解像度が異なる端末の画面の場合でも，大きさを揃えて比較を行うことができる．

図 5 矩形領域抽出の例

Step2 では，以下のような更に細かい Step で正解画面においてなるべく画面要素に近い単位で矩形領域の抽出を試みる．

- Step2-1：画面中の文字列が，個別の文字ではなくなるべく繋がった文字列の矩形として抽出できるように，Sobel フィルタを用いて横方向に階調差が大きい領域を，文字列の可能性が高い領域として抽出し，この領域へ横方向のブラー処理をかける．
- Step2-2：正解画面全体に対し Canny 処理を行いエッジを検出する．
- Step2-3：Suzuki85 のアルゴリズムを活用してエッジから輪郭となりえる頂点のグループを全て抽出する．
- Step2-4：輪郭を形成する頂点グループそれぞれに対して，頂点を囲む凸包をつくり，その凸包を囲む長方形をを画面要素相当の矩形領域として抽出する．

図 5 に，このような手順で正解画面から矩形領域を抽出した例を示す．

Step3 では，正解画面における矩形領域が，比較対象画面におけるどの領域に相当するかを特定するため，正解画面と比較対象画面における AKAZE 特徴量の抽出とマッチングを行う．特徴ベースマッチングでは，画像からエッジやコーナー等の特徴点を抽出し，その周囲の領域から特徴量を計算することで画像間のマッチングを行う．そのため，正解画面と比較対象画面間の解像度差などにより，画面内の画面要素の個別の拡縮率が多少変わるなどした場合でも，拡縮率の差異を吸収してマッチングできる．図 6 に特徴量の抽出とマッチングを実際に行った例を示す．図 6 では，正解画面と比較対象画面で対応が取れた特徴点同士を線で結んだものを示している．

Step4 では，Step3 の特徴量マッチング結果を用い，正解画面における各矩形領域に対応する比較対象画面の矩形領

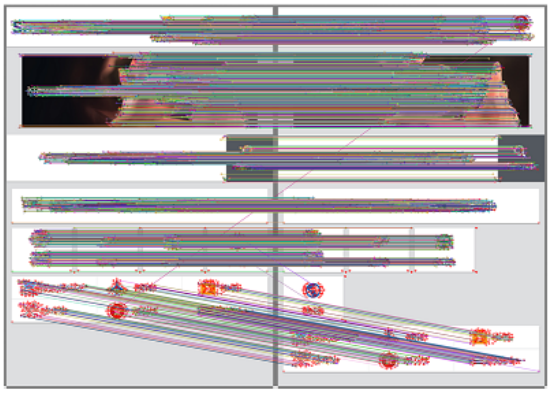

図 6 特徴量の抽出とマッチングの例

域を特定し矩形ペアを作成する．具体的には，正解画面のある矩形 X に対し，X 内に特徴点があった場合，「矩形 X の左上の座標点」-「正解画面側の特徴点座標点」で計算される 2 次元ベクトル V を算出し，「比較対象画面側の特徴点座標 」$+ V$ を計算することで比較対象画面側の対応矩形領域の左上の座標点を算出し，対応矩形領域を特定することで，正解画面側と比較対象画面側の矩形ペアを作成する．また，正解画面，比較対象画面，それぞれの対応する矩形領域内の特徴点間の距離の情報も活用し拡縮のある矩形についても矩形ペア (拡縮率も保持させる) を作成する．

Step5 では，矩形ペアでマッチングの度合いを計測し一致/不一致を判定する．一致かどうかの判定はテンプレートマッチングを用いて，閾値を超えたかどうかで一致かどうか判定する．このとき，拡縮のある矩形ペアでは拡縮率を考慮して縦横の画素数を揃えてリサイズした上でテンプレートマッチングを行う．テンプレートマッチングの結果，不一致だった場合は差異理由として「変更」のタグを付与する．また，拡縮率が 1 でない場合には差異理由として「拡縮」のタグ（及び拡縮率の値）を付与する．正解画面側の矩形で特徴点が存在せず対応がとれなかったものは差異理由として「消失」というタグを付与し，正解画面側の矩形と比較対象側の矩形の座標位置が違えば「移動」のタグ (及びどれだけの画素数移動したかの情報) も付与する．最後に，比較対象画面で Step1 と同様の矩形抽出を行い，正解画面との矩形ペアが取れていない領域で矩形が抽出された場合には，その矩形を差異をとして抽出し，差異理由として「追加」のタグを付与する．

3.2 (2) テスト結果判定機能

図 4(2) テスト結果判定機能では，(1) レイアウト差異検出機能で抽出した差異をその差異の理由と伴に一覧でテスト担当者へ掲示する．図 1 で示した正解画像と比較対象画像を例にとり，テスト結果判定機能のイメージを図 7 に示す．

差異一覧は図 7(ア) のように理由とともに表示される．また，テスト担当者が図 7(ア) で選択した差異は図 7(イ) のように正解画面上と比較対象画面上でハイライト表示されるため，視覚的にどの領域の差異かどうかを確認することができる．図 7 では，No.7 の差異を選択しており，これは図 2(2) の差異 (アイコンが存在しない) に相当する．テスト担当者はこれを目視で確認することで OK/NG を判定し，図 7(ア) で OK/NG のチェックをつけ，NG であった場合はその理由を図 7(ウ) へ入力する．また，テスト担当者は「移動量が xx ピクセルまでは許容」，「拡大縮小が yy% までは許容」といった合否ルールを指定することができ，このルールの条件にあてはまる差異に関しては自動で OK の判定を行い，人が確認するときにスキップすることも可能である．

このように，差異を漏れ無く抽出したものを表示し，それをテスト担当者がひとつひとつ確認していくことができる．また，テスト担当者が効率よくこのようなチェック作業を行えるよう，操作画面ではショートカットキーで大部分の操作が行えるよう工夫している．これらにより，テスト担当者は効率よく，UI レイアウト崩れを検出していくことができる．

3.3 (3) レポート出力機能

図 4(3) レポート出力機能では，テスト担当者が判定した OK/NG の情報と，NG 箇所に相当するアプリケーション画面上の差異の箇所の情報を使ってレポートを作成する．レポートでは，正解画面，比較対象画面上において NG と判定された差異の箇所へ通し番号をふり，その通し番号を近くに表示した上でハイライトし，各番号に対応する NG 理由のコメントの一覧についても出力する．これにより，テスト担当者は自動生成したバグレポートを開発担当者にそのまま渡すことができ，手動でレポートを作成していた労力も削減することができる．

4. 評価

4.1 観点

本研究では，これまでテスト担当者が完全に目視で確認を行っていたときの稼動を削減し，UI レイアウト崩れの検出漏れを減らすことを目指している．そのため，以下の観点で評価実験を行った．

- 提案手法が全ての UI レイアウト崩れを差異として検出できているか．
- 完全手動で目視確認した場合と比較し，提案手法を使って確認した場合に「誤り発見率」と，「確認に要する時間」がどれだけ改善されるか．

これに加え，提案手法におけるテスト結果判定機能を使った場合と，完全手動の目視確認の両方を行ってもらったテスト担当者にヒアリングを行い，提案手法の良い点，使いにくい点を列挙してもらうことも行った．

ⓒ 2016 Information Processing Society of Japan

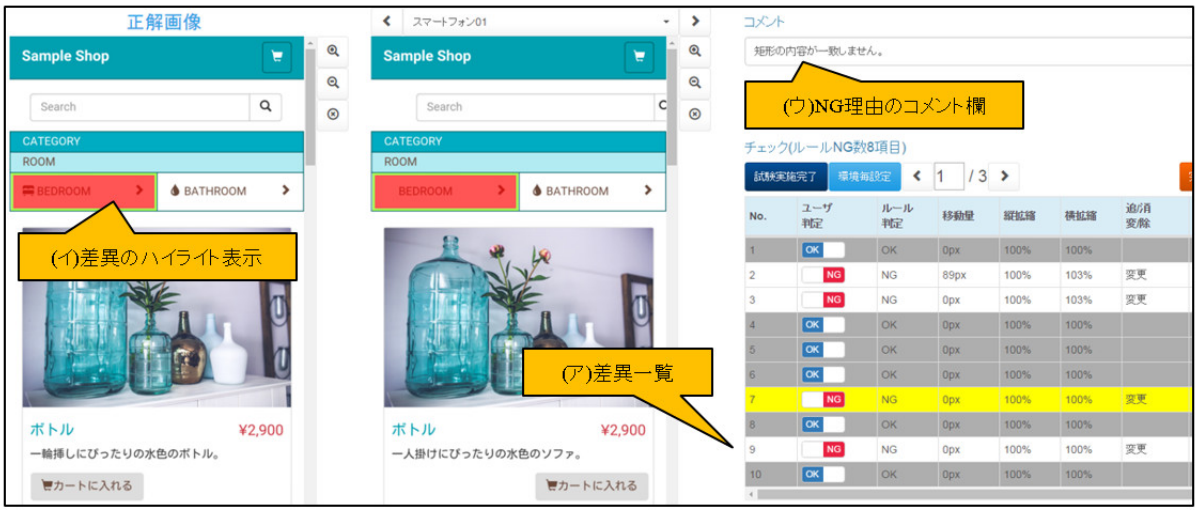

図 7 テスト結果判定機能のイメージ

4.2 方法

Web アプリケーション画面のスクリーンショットをとり正解画面を用意し，その正解画面へ差異 (OK と判断すべき差異，NG と判断すべき差異の両方を含む) を人為的に埋め込むことで，UI レイアウト崩れが入っている比較対象画面を作成し，これを用いて前節の観点で評価を行う．埋め込む差異の種類は以下のようにした．

- 画面要素の追加，削除，変更，移動，拡縮といった基本的な差異
- 改行位置の変更，画面要素同士が重なり合う，フォント違いといった実際によく起こりうる差異

これらの差異を 1 画面 (縦長でスクロールを要する画面) に対して約 10 個〜20 個（このうち，NG と判定すべき差異は 0〜最大 15 個程度) をランダムに埋め込むことで，正解画面と差異が埋め込まれた比較対象画面のペアを実験データとして作成した．そして，提案手法によりこの実験データに埋め込まれた誤り (UI レイアウト崩れ) を差異として検出できるかどうかの確認を行う．加えて，開発経験のある被験者に，完全手動で目視確認した場合と，提案手法を使って確認した場合の 2 通りの方法でテスト結果確認を行ってもらい，誤り発見率と確認に要する時間の 2 つを測定した．完全目視の確認については，なるべく効率よく行ってもらえるよう，エクセルシートに正解画像と比較対象画像を並べて貼り付けファイルをあらかじめ用意し，被験者にはシートを見てもらいながら差異の確認を行ってもらった．ヒアリングについては今回の実験の被験者全員に対して行った．

4.3 実験結果

提案手法は，埋め込んだ誤りの箇所を全て差異として検出できた．また，提案手法を用いた場合と，手作業 (完全目視確認) の場合における誤り発見率と，確認に要した時間を図 8 に示す．誤り発見率は提案手法で平均 88.9% ， 手作業では平均 87.6% とほぼ同じであった．また，確認に要した時間は提案手法では平均 13.0 分，手作業では平均 28.7 分となり 45% まで削減できた．

4.4 ヒアリング結果

ヒアリング結果について，多かったコメントについていくつか抜粋して紹介する．良い点としては以下のようなコメントがあった．

(1) どのような差異であるかの理由が確認できる点．例えば，小さな移動，拡大縮小は理由欄の数値を見ればわかるため問題ないものはすぐに OK とできる．
(2) 比較箇所の差異の位置がハイライトされたり，自動で差異の箇所へ画面がスクロールする点．
(3) 慣れは必要だが，キーボードだけで次々とチェックしていける点．
(4) 直感的に UI レイアウト崩れの位置がわかり，NG 判定をした差異の箇所に通し番号が振られたレポートが出力される点．

使いにくいと感じた点としては以下のようなコメントがあった．

(1) 正解画面と比較対象画面を横に 2 つ並べて表示しているため，画面要素が小さく表示されてしまい，比較しずらい．
(2) 大きな矩形領域が「変更」と掲示される場合があり，矩形領域内のどこに変更があるのかがわかりにくい場合がある．
(3) ひとつの UI レイアウト崩れに対して複数箇所が差異として検出されることがある．
(4) UI レイアウト崩れとする必要のない差異も多く検出されるため，いちいち確認するのは手間がかかる．
(5) 差異のある箇所は抽出しているが，マッチング先が

ⓒ 2016 Information Processing Society of Japan

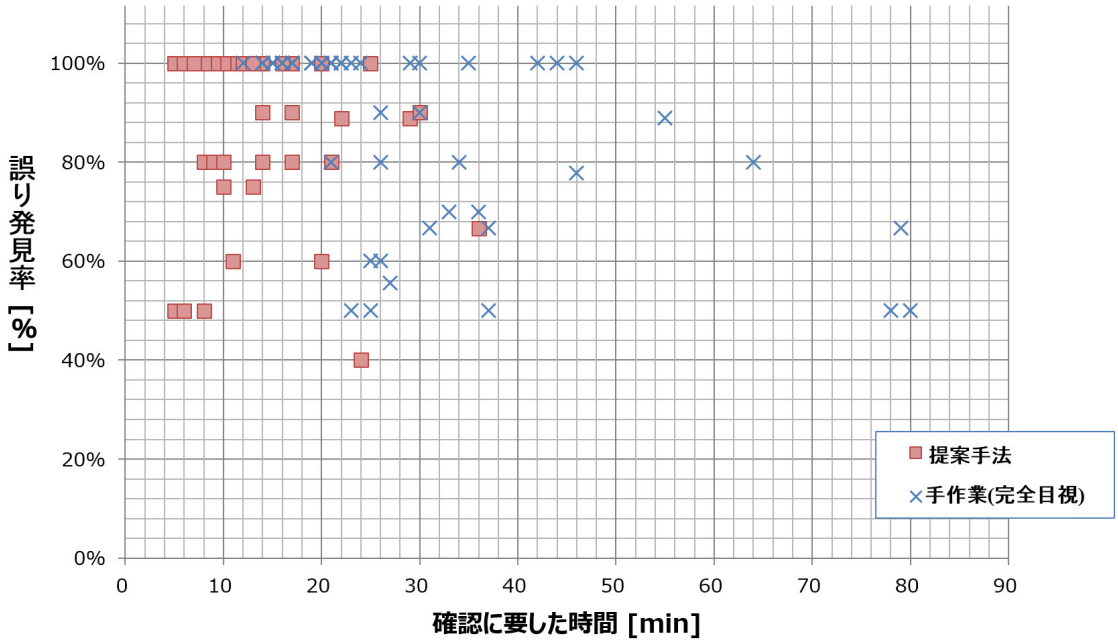

図 8 評価結果

誤っていることがある．

4.5 考察

ヒアリングにおける良いと感じ点のコメントに関して，差異へ理由も付与してテスト担当者へ掲示することが，差異が UI レイアウト崩れであるかどうかの判断に役立っていることがわかった (コメント (1))．また，テスト結果判定機能の UI についても差異のハイライトや，ショートカットキーで効率よく操作できる点など好評だった (コメント (2)〜(4))．これらにより，テスト担当者は目視確認に比べて短時間で確認作業を終えることができたと考えられる．

図 8 で，提案手法を用いた場合でも誤り発見率が 100% になっていない場合がある理由は，提案手法はあくまで差異を全て自動検出するところまでを行っており，最終的にその差異が OK か NG かの判断は人によって目視確認で行われ，漏れが生じる可能性があるためである．提案手法を用いた場合でも完全目視の確認に比べて誤り発見率が向上しなかった原因は，ヒアリングにおける「提案手法が使いにくいと感じた点」に起因すると考えられる．

提案手法が使いにくいと感じた点に関して，(1) についてはテスト結果判定機能においてフルスクリーンで正解画面と比較対象画面を並べて比較できるようにしたり，もしくはより大きなディスプレイを使うようにするなどすれば改善可能である．(2) については，例えば抽出した矩形領域同士で画素単位の差異をとりそれを表示することで，矩形領域内の細かい差異の箇所がわかりやすくできる可能性がある．(3) に関しては，人が見たときにわかりやすい単位の粒度で差異が検出できていないことが一因と考えられ，より画面要素として意味のある単位の矩形へ複数の矩形を統合するなどの工夫をしていく必要がある．また，(4) については，より人の判断に近づけた自動判定を行い，UI レイアウト崩れと無関係の差異を無視できるようにすれば解決できる可能性がある．今回の提案手法では「移動」，「拡縮率」といった差異については合否ルールで微小な差異について無視できるように設定できたが，更にこのような仕組みを高度化していく必要がある．例えば，機種間で文字フォントが異なる場合，文字列は同じでもフォントが違い画像として異なるため，提案手法では差異として抽出する．しかし，フォントの差異は UI レイアウト崩れではないため，このような差異を自動判別し無視できるようになれば，テスト担当者は余計な差異を確認する必要がなくなり，より効率よく確認作業が行えるはずである．

(5) については，特徴量を用いたマッチングでは矩形の対応付けは厳密ではなく「似たような特徴をもつ領域」として対応付けるため，一定数は発生してしまう．これを解決するためには，例えば画面要素間の相対的な位置関係の情報を用いるなど，より多くの情報を用いて正確な対応付けを行っていく必要がある．

ⓒ 2016 Information Processing Society of Japan

5. 結論

画面要素の単位で正解画面と比較対象画面の差異を理由付きで自動抽出し，それらの差異をテスト担当者へ掲示しOK/NG の判定を行ってもらい，最後にその判定結果をバグレポートとして生成する手法を提案した．提案手法している手法は，通常，テストで実施証跡として記録されるアプリケーション画面のスクリーンショットのみを入力としており他の入力資材は不要であるため，アプリケーションの実装技術に依存せずにテストで活用することが可能である．

提案手法では矩形抽出，特徴量抽出とマッチングなどの画像処理の技術を活用し，正解画面と比較対象画面における矩形領域を対応付けて，その差異を抽出するという工夫をしている．また，対応付けられた矩形のペアについては場合は移動量，拡縮率等の情報も計算しておくことで，これらの情報も含めて，2 つの画面の差異を画面要素の単位で出力する．

今後は，評価実験で得られたフィードバックを元に，テスト担当者がより効率よく UI レイアウト崩れの検出を行えるよう，提案手法の改善を進めていきたい．

参考文献

[1] DiffImage. https://sourceforge.net/projects/diffimg/.
[2] FightingLayoutBugs. http://code.google.com/p/fighting-layout-bugs/.
[3] さまざまな端末でアプリ画面の間違い探しを簡単に. NTT R&D フォーラム パネル, 2 月 2016. https://labevent.ecl.ntt.co.jp/forum2016/elements/pdf_jpn/03/S-10_j.pdf.
[4] さまざまな端末でアプリ画面の間違い探しを簡単に. NTT R&D フォーラム 紹介動画, 2 月 2016. https://www.youtube.com/watch?v=OUeAz-iBLuY.
[5] Abdulmajeed Alameer, Sonal Mahajan, and William G. J. Halfond. Detecting and localizing internationalization presentation failures in web applications. In *2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, IL, USA, April 11-15, 2016*, pp. 202–212, 2016.
[6] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, Vol. 41, No. 5, pp. 507–525, 2015.
[7] S. Roy Choudhary, H. Versee, and A. Orso. Webdiff: Automated identification of cross-browser issues in web applications. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pp. 1–10, Sept 2010.
[8] Adrian Kaehler Gary Bradski. 詳解 OpenCV : コンピュータビジョンライブラリを使った画像処理・認識. オライリー・ジャパン, 2009.
[9] Akihiro Hori, Shingo Takada, Haruto Tanno, and Morihide Oinuma. An oracle based on image comparison for regression testing of web applications. In *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, pp. 639–645, 2015.
[10] S. Mahajan and W. G. J. Halfond. Detection and localization of html presentation failures using computer vision-based techniques. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1–10, April 2015.
[11] Sonal Mahajan and William G.J. Halfond. Finding html presentation failures using image comparison techniques. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pp. 91–96, New York, NY, USA, 2014. ACM.
[12] Shauvik Roy Choudhary, Mukul R. Prasad, and Alessandro Orso. X-pert: Accurate identification of cross-browser issues in web applications. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pp. 702–711, Piscataway, NJ, USA, 2013. IEEE Press.
[13] Thomas A. Walsh, Phil McMinn, and Gregory M. Kapfhammer. Automatic detection of potential layout faults following changes to responsive web pages (N). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pp. 709–714, 2015.
[14] 池之上あかり, 中野直樹, 藤澤正通. 分割した web ページキャプチャ画像を用いた画像差分検証手法の提案. ソフトウェアテストシンポジウム 2016 予稿集, pp. 75–81, 3 月 2016.

ⓒ 2016 Information Processing Society of Japan