研究コース 5（GUI-Test チーム）

GUI 要素の階層構造構築・比較による
視覚的回帰テスト差分検出方法の実証評価

研 究 員：塚越　章弘（株式会社日本科学技術研修所）
主　査　：石川　冬樹（国立情報学研究所）
副 主 査：栗田　太郎（ソニー株式会社）
　　徳本　晋（株式会社富士通研究所）

研究概要
既存の GUI アプリケーションを改修する際に，不具合が混入していないことを担保し，かつ効率的な開発を進めるため，回帰テストの自動化を行うことがある．視覚的回帰テストにより GUI 画面が正しく表示されていることを確認できるが，何らかの原因により画面内 GUI 要素のズレが存在した場合，本質的でない差分を検出してしまい，結果として開発効率を下げてしまうことがある．このような問題を解決するため，本研究ではコンピュータ・ビジョンのライブラリを活用することとした．検証の結果，本手法の有効性を確認することが出来た．

1.　はじめに・研究の背景

ソフトウェア開発において，既存のアプリケーションに対して新規機能を付加したり，パフォーマンス改善を行うため，改修作業が行われることが良くある．また，保守性を向上させるためにソースコードのリファクタリングが行われることもある．このような修正作業が行われた場合に，目的外の機能に新たなバグを埋め込んでしまう，いわゆるエンバグが発生することを防ぐため，回帰テストが良く利用されている．アプリケーション画面についても視覚的回帰テスト（VRT:Visual Regression Test）により，スクリーンショット画面が正しく表示されていることをテスターが確認することが出来る．VRT では，古いバージョンと新しいバージョンの画像をピクセル単位で比較を行い，違いのあるピクセルの強調表示等が行われる．これにより，GUI アプリケーションに対しても無影響確認の自動化を実現でき，開発効率を上げることが期待できる．

しかしながら画面内 GUI 要素に微小なズレが存在した場合，差分画像に余計な差分が入り込んでしまい，テスターが本来検出すべき，アプリケーション不具合による差分を確認することが難しくなってしまう．これを解決するため，スクリーンショット画像内の GUI 要素をエッジ検出し，各要素間の階層構造を解析・ツリー構造の構築，比較を行う方法によって，差分結果を取得する方法を提案する．

2.　現状分析・課題

2.1　現状の視覚的回帰テストについて

VRT は新旧バージョンの画像を比較する手法で，スクリーンショット画像を用意すれば利用することが出来，OS やブラウザ等に依存せずに幅広く適用できることが特徴に挙げられる．この手法を利用したツールとして，DiffImg[1]や BlinkDiff[2] などがある．

しかし VRT では，スクリーンショット画面の比較において，単純なピクセル単位の差分画像や平均差分画素数の算出等を行うため，画面内 GUI 要素のズレが存在した場合には，画面全体的に差分箇所が検出されてしまい，改めて手動で比較結果を確認しなければならなくなる．結果として無影響確認の自動化を阻害する要因となってしまう．

2.2　課題

単純なピクセル単位差分比較の問題点として，以下のようなものが挙げられる．
・画面キャプチャ取得時の精度により，微小なズレが入り込むことがある
・画像内の要素にズレが入り込むと，差分画像に大きく影響してしまう
・修正前後で実行環境を完全に合わせることが出来ればズレは発生しないが，データベース連携，レジストリを多数扱うアプリケーションの場合，毎回環境を併せることは非常に工数が掛かってしまう
・GUI コントロール（GUI 要素）を動的に作成するアプリケーションで，レイアウト配置にズレが入り込むことがある（図 2-1，図 2-2）．ここで条件によって動的に内容が変化する領域を動的領域と呼んでいる．

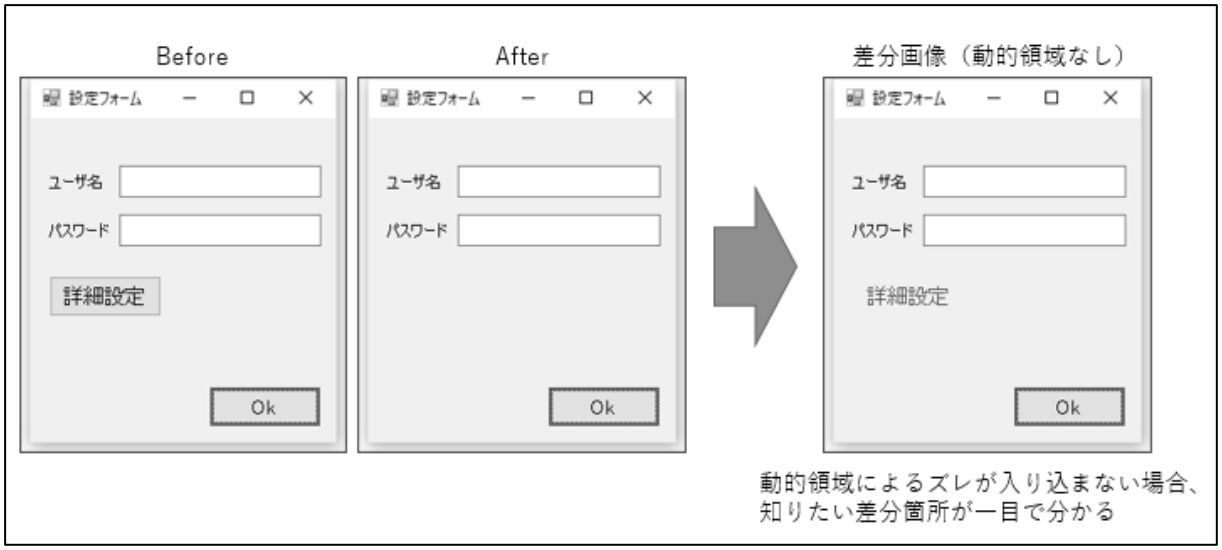

図 2-1 差分画像取得結果（動的領域がない場合）

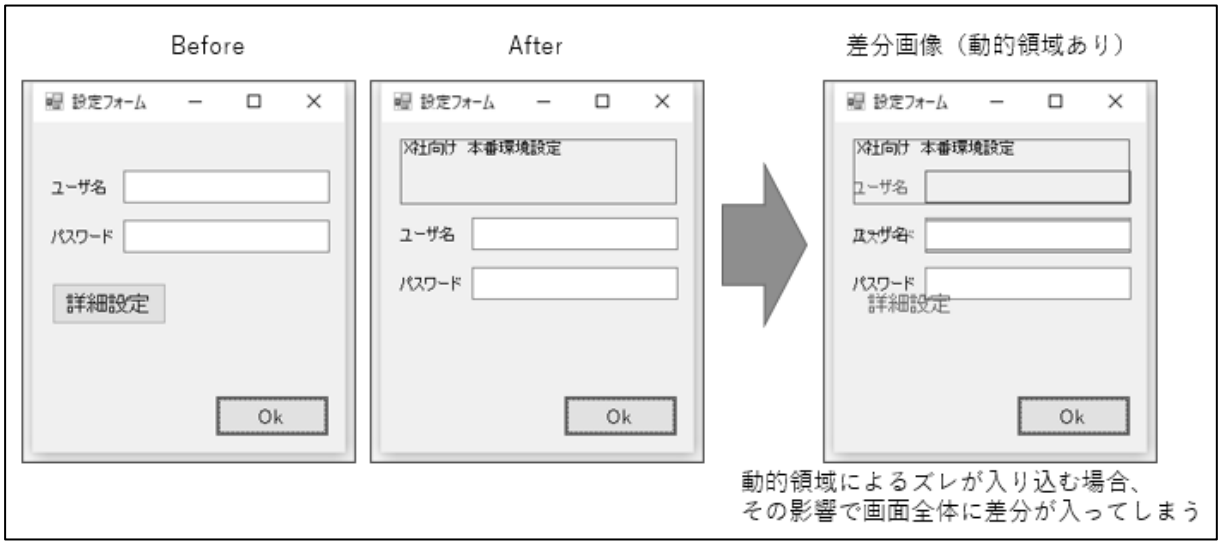

図 2-2 差分画像取得結果（動的領域がある場合）

3. 先行研究の調査

本研究を行うにあたって，先行している研究の調査を行った．その結果，Region ベースの差分検出器（ReBDiff:Region-based Difference detector）を使用した，意味のある差分を検出する方法[3]や，差異があっても問題ない領域を差異として検出しないように動的領域にマスキングを掛け，比較対象から除去する方法[4]などが提案されている．

これらの手法の多くは,適用対象を Web アプリケーションに対するクライアント画面(ブラウザ）としているものとなっていた．Web アプリケーションの場合，HTML のデータ構造を DOM（Document Object Model）として取得することが出来，Selenium[5]などを使用して自動操作しやすく，比較を行うためのスクリーンショット画面を取得しやすいといったことが考えられる．

本研究では，ReBDiff での実装ロジックの着想にヒントを得て，さらに適用対象をスタンドアロン型の GUI アプリケーションに対しても同様に差分検出が出来るロジックの提案，検証を行う．また本研究では，微小なズレが入り込む要因となる動的領域として，ブランク領域のみ取り扱う．

4. 解決策の提案

4.1 差分検出の処理概要

OpenCV[6]や Google Cloud Vision API[7] を利用して，本研究の画像解析に使用することとした．OpenCV は画像処理・画像解析および機械学習等の機能を持つオープンソースのライブラリで，Google Cloud Vision API は Google Cloud Platform が提供するサービスでラベルのテキスト情報等も読み取ることが出来る．これらを組み合わせて，GUI 要素間の階層構造を解析・ツリー構造の構築，比較を行い，差分検出を試みることとした．処理概要は表 4-1 のように行っている．

表 4-1 差分検出の処理概要

	処理内容
Step1	新旧バージョン画像の差分画像を取得し,差分画素数が閾値以下で一致とみなすことができたら，差分なしと判断する．差分ありの場合，step2 以降を継続する．
Step2	旧バージョン画像をグレースケール（RGB のカラー画像に対して，白黒の濃淡を表現した画像）で読み込む
Step3	Step2 で生成したオブジェクトに対して,OpenCV の画像閾値処理を行うメソッドを使用し，二値画像化を行う．この step でオブジェクト内要素の輪郭を強調することが出来る．
Step4	Step3 で生成したオブジェクトに対して,OpenCV の輪郭を検出するメソッドを使用し,全体画像内に含まれるすべての画像要素を取得する．この Step で GUI 要素を拾い出すと同時に，各要素間の階層構造も取得することが出来る．
Step5	輪郭を持たない GUI 要素としてラベル要素も存在するため,それを補完する方法として，Google Cloud Vision API を利用し，差分比較の際に補足情報として利用する．ラベルのように輪郭を持たない要素は，Web アプリケーションの DOM の場合と異なり，データ構造として取得することが難しいため，このような補完処理を行っている．
Step6	新バージョン画像に対しても step2～5 と同様の処理を行う．新旧バージョン画像に対する GUI 要素ツリーの比較を行い，差分比較を行う．ここで２つのツリー間で対応する要素のペアを見つけ出し，それぞれの差分画像を求める．

4.2 処理の詳細

4.2.1 二値画像化ついて

表 4-1 Step3 では，ある画素値が閾値よりも大きければ白，そうでなければ黒のように処理する二値画像化を行っている．この処理を行うことにより，後続処理でオブジェクトの輪郭を捉えることが出来るようになる（図 4-1）．二値画像化も多様な方法が存在しているが，適応的二値化処理という，全体的な閾値を固定せずに，注目画素とその周囲にある

画素の平均値を閾値とする方法がある．GUI 要素には様々なパターンの配色が設定される可能性があるため，本研究では適応的二値化処理を採用することとした．

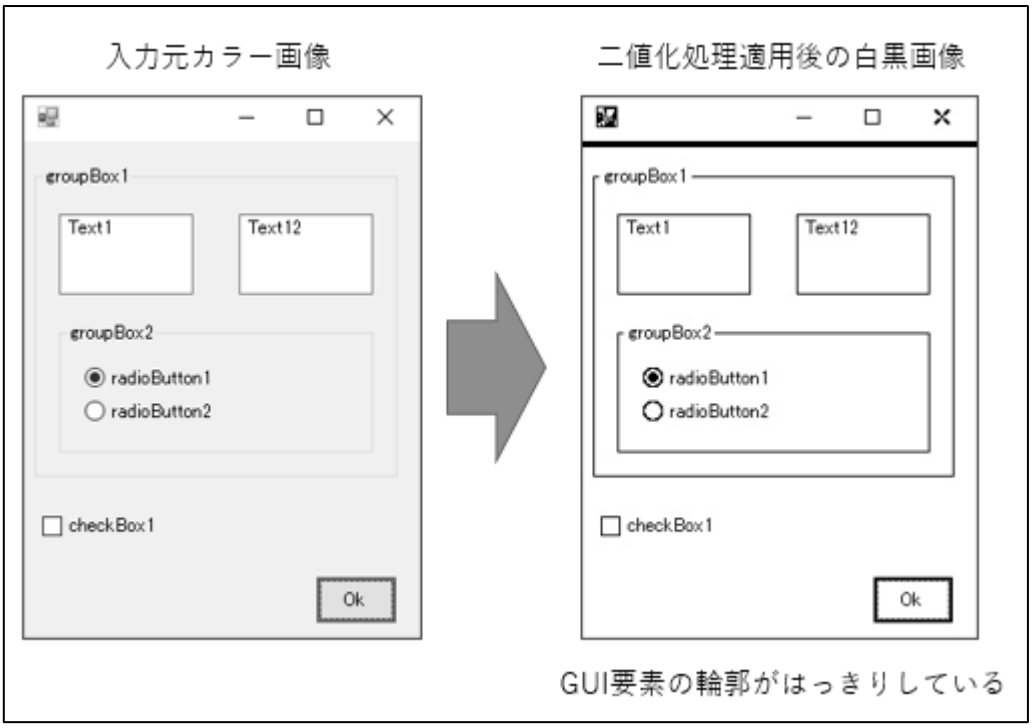

図 4-1 二値化処理の適用

4.2.2 要素間の階層構造について

表 1 Step4 では，画面全体に含まれる GUI 要素の検出を行い，同時に階層構造も取得している．GUI 要素が入れ子のように存在している場合，それを階層構造を持つものと表現している．例えば図 4-2 のようなレイアウトになっていた場合，一番外側の輪郭が階層 0，その一つ内側に含まれる輪郭が階層 1， さらにその一つ内側に含まれる輪郭が階層 2 のようになる．また，階層 0 と階層 1 は親子関係となっており，階層 0 に位置する要素が親要素，階層 1 に位置する要素が子要素となる．その配下の階層も同様に親子関係を持つ．階層 2 のように子要素が複数になる

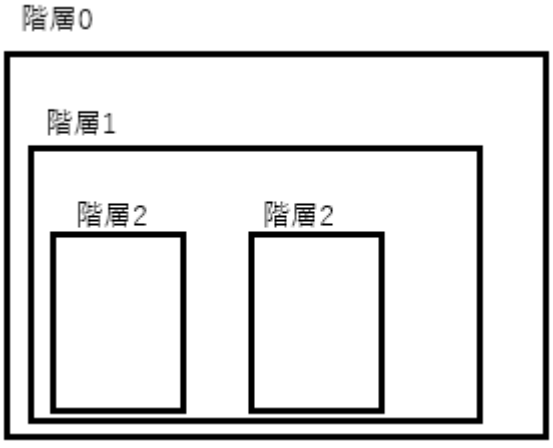

図 4-2 GUI 要素の階層表現

Step4 の処理で，検出された輪郭（GUI 要素）はそれぞれ識別番号・階層情報・親子関係・

座標情報・面積等を持ち，それらをツリー構造として取得することが出来る（図 4-3）．図 4-3（左）では，輪郭が検出されていることを確認するため，検出された輪郭の四隅をプロット表示している．

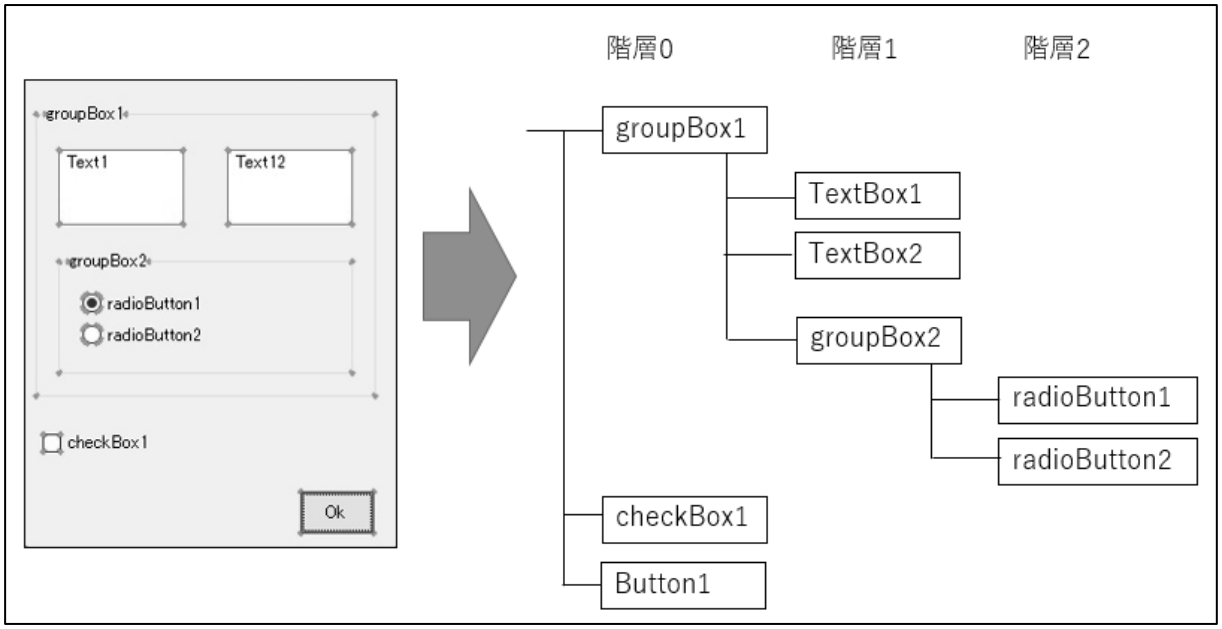

図 4-3 GUI 要素のツリー構造

4.2.3 輪郭を持たない GUI 要素への対応について

表 4-1 Step5 では，ラベル要素のように輪郭を持たない GUI 要素も差分比較できるようにするため，Google Cloud Vision API のテキスト検出メソッドを利用し，検出した Bounding Box を GUI 要素として扱っている．

4.2.4 要素ごとの差分比較について

表 4-1 Step6 では，取得した新旧画像の GUI 要素ツリーを比較し，ツリーのルートから辿っていき，階層位置や面積情報等を元に対応する GUI 要素のペアを紐づけていく．そして対応する GUI 要素ペアごとに差分比較を行う．

5. 解決策の評価，考察

5.1 検証環境

以下に示す環境にて動作検証を行った．

OS:Windows10

検証対象プログラム：C# Windows Form アプリケーション

検証を行うプログラム：Python3.7 + OpenCV4.4 + google-cloud-vision2.0

検証対象プログラムには，一般的に良く使用される GUI 要素を配置したアプリケーションを用意した．

5.2 検証方法

テストケースとして， GUI 要素の追加・消失がなく，GUI 要素間の位置関係に微小なズレのみが存在する場合を想定したテスト対象画面を用意し，評価を行った．また，GUI 要素の種類としては，一般的によく使用されている Button，CheckBox，ComboBox，LixtBox，Scrollbar，Label，GroupBox，TextBox，RadioButton を使用することとした．テスト対象画面の例を図 5-1 に示す．

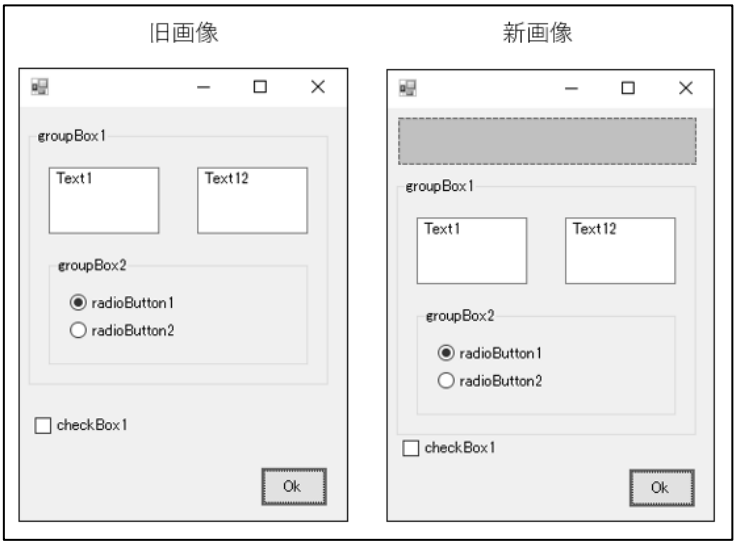

図 5-1 テスト対象画面の例（新画像上部に微小なズレが存在）

評価ポイントとしては,
・評価項目 A：画面内の全 GUI 要素を検出できているか
・評価項目 B：取得した GUI 要素によるツリーが適切な階層構造で構築できているか
・評価項目 C：対応する GUI 要素間で適切に差分比較が行われているか
としている.

5.3 評価結果

評価結果は表 5-1 の通り，No5 の ScrollBar が存在するケース以外はすべて問題なく合格している.

表 5-1 評価結果

No	テスト対象画面	評価項目 A	評価項目 B	評価項目 C
1	1 つ以上の Button が存在	○	○	○
2	1 つ以上の CheckBox が存在	○	○	○
3	1 つ以上の ComboBox が存在	○	○	○
4	1 つ以上の ListBox が存在	○	○	○
5	1 つ以上の ScrollBar が存在	○	○	×
6	1 つ以上の Label が存在	○	○	○
7	1 つ以上の GroupBox が存在	○	○	○
8	1 つ以上の TextBox が存在	○	○	○
9	1 つ以上の RadioButton が存在	○	○	○

差分比較の評価において,
・微小なズレに対してはアプリケーション不具合による差分として検出されない
・意図的に設定した不具合箇所については，差分として検出される
ことを合格としている.

また，No5 の ScrollBar が存在する場合に失敗する原因として，GUI 要素の一部分が欠落してしまうためとなっている.（図 5-2）

図 5-2 GUI 要素に Scrollbar が存在する場合の画面例（右下ボタンの一部が欠落）

5.4. 評価結果の考察

表 5-1 に示す通り，ほとんどの場合うまく機能している．以下，うまくいかなかった場合について考察を進める．

No5 の ScrollBar による GUI 要素欠落が発生している場合については，画面キャプチャ時に少なくとも全要素が表示されるようにする工夫を入れないと，それ以降の対応は難しいと思われる．

また，画像内文字列の一部や模様などを，独立した細かいオブジェクトとして検出してしまうことが見受けられた．例えば文字列「checkBox1」の文字「B」のような囲みのある領域も輪郭検出していることがあった（図 5-3）．これを防ぐため，ある閾値よりも面積が小さいオブジェクトは検出対象外となるようにしたところ，問題なく GUI 要素の輪郭を検出し，差分比較が行えるようになった．この閾値よりも小さい GUI 要素を差分比較対象とする必要がある場合は，さらに何らかの工夫が必要となる．

図 5-3 文字「B」の囲み部分が輪郭検出された画面例

また，GUI 要素の境界線をデフォルト設定で差分比較を実施した場合は特に問題は発生していないが，デフォルト設定よりも数ピクセル程度以上太くした状態で階層構造を構築しようとした際，境界線の外側と内側に複数の輪郭を検出し，冗長な階層作成が見受けられた．この状態でも問題なく差分比較を実施できるものの，冗長な出力結果となる．この冗長構成を解消するには，ほぼ同一位置に存在し，面積もほぼ同じ要素は同一視するような処理等が必要になる．

また，差分比較対象の「微小なズレ」が対象プログラムの仕様によってはレイアウトの不具合となっている可能性もある．微小かどうかを仕様に合わせて決めることが出来るようにすると良いと考えられる．そこで，比較を行う際の外部パラメータとして，どの程度の距離までを微小と見なすかといった閾値設定を設ける等の工夫が必要と思われる．

6. まとめ

本研究で提案している解決策を検証した結果，画面内 GUI 要素に微小なズレが存在した場合でも，それをアプリケーションによる不具合と誤検知せず，意図的に設定した不具合箇所については，差分として問題なく検出されることが確認できた．

ScrollBar による画面欠落の課題はあるものの，今後本研究での差分比較方法を実際の運用フェーズへ組み込み，実用に耐えられるか検証する価値はあるものと考えられる．

また，本研究の解決策として使用したデータ数はまだ十分とは言えず，今後実際に使用されている GUI プログラムのレイアウトを参考にしながら，より多くのテストパターンで検証を積み重ねていく必要があると考えられる．

テスト自動化においては,大量のテストが一気に行われることが想定される.そのため，大量の差分比較結果の一覧をテスターが確認した際，本手法のテスト結果を効率よく把握できるように，レポート出力の工夫も必要となることが考えられる．

参考文献

[1] DiffImg, https://ja.osdn.net/projects/sfnet_diffimg/

[2] BlinkDiff, https://github.com/yahoo/blink-diff

[3] Haruto Tanno, Yu Adachi, Yu Yoshimura, Katsuyuki Natsukawa, Hideya Iwasaki, 「Region-based Detection of Essential Differences in Image-based Visual Regression Testing」, Journal of Information Processing Vol.28 268-278(Apr. 2020)

[4] 安達 悠, 丹野 治門, 吉村 優,「Visual Regression Testing において画面要素の位置関係によってマスク領域を指定する手法」, https://doi.org/10.11309/jssst.36.4_53

[5] Selenium, https://www.selenium.dev/

[6] OpenCV, http://opencv.org/

[7] Google Cloud Vision, https://cloud.google.com/vision?hl=ja