

テストスクリプトに対するミュートーションと 画面遷移の比較に基づいた Web アプリケーション 異常系テスト支援手法の提案

山下 智也 阿萬 裕久 川原 稔

近年、Web アプリケーションの普及に伴い、Web アプリケーションに対するテスト自動化のニーズも高まってきている。Web アプリケーションのテストでは、さまざまなテストシナリオの下で Web ブラウザを操作する必要がある。これについては Selenium を使った操作の記録と自動実行が有益な技術として広く知られている。一方、そういった自動化支援は、あくまでも人手による再実行が不要になるだけであり、テストケースを自動生成してくれるわけではない。そこで本論文では、正常系テストケース（テストスクリプト）に対してミュートーション技術を適用し、そこから変異テストスクリプトを自動生成・自動実行し、動作確認を支援する目的で、Web ブラウザ上での画面内容を自動収集し、結果として得られる画面と正常系テストの画面の差分に基づいて想定外の結果が得られることがないか確認する手法を提案している。そして、事例研究を通じて提案手法の有効性について検討している。

1 はじめに

今日、Web ブラウザをインタフェースとする Web アプリケーションは、業務システムから消費者向けサービスに至るまで広く利用されており、その品質確保のために、さまざまな動作シナリオの下で Web アプリケーションの網羅的な動作テストを行うことが重要となっている。Web アプリケーションの動作検証では、Web ブラウザを通じたユーザインタフェースの操作が必要となるため、仕様通りの動作を確認する正常系テストだけでなく、例外的な条件や入力エラーなどを想定した多種多様な異常系テストも網羅的に行う必要がある。しかしながら、このような多様なシナリオを手手で設計し、正確に実行・管理するこ

とは非常に困難であり、工数や品質の両面に課題が残る。とりわけ異常系シナリオでは、操作の組合せが爆発的に増加するため、人的リソースによる十分な検証には限界がある。

このような背景から、Web アプリケーションにおけるテストの自動化は極めて重要であり実際に Selenium [1] [3] 等の技術を用いることで、人手で行ったブラウザ操作をいったん記録しておけば、同じ操作を自動的に再実行できるようになっている。それゆえ、記録された正常系や異常系のテストシナリオを一度実行しておけば、2 回目以降の再実行を容易に自動化できる点で、大きな効果を発揮している。しかしながら、それはあらかじめ人手で用意したテストを自動実行できるだけであって、テストケースを自動的に“生成する”わけではない。特に異常系テストシナリオの場合、さまざまな“例外や操作ミス”を想定することになるため、その多様性ゆえに人手で用意するには限界がある。

そこで本論文では、Web アプリケーションのため

Tomoya Yamashita, 愛媛大学大学院理工学研究科, Graduate School of Science and Engineering, Ehime University.

Hirohisa Aman, Minoru Kawahara, 愛媛大学総合情報メディアセンター, Center for Information Technology, Ehime University.

のテスト自動化支援の一環として、異常系テストケース支援手法を提案する。具体的には、Selenium を用いて記録された正常系テストケース（ブラウザ操作）を Python 形式のテストスクリプトとして取得し、そのテストスクリプトに対してプログラム解析並びにミューテーション操作を施すことで多様な変異テストケースを自動生成・実行する手法を提案する。

さらに、本手法では自動生成された変異テストケースそれぞれの自動実行結果に対する確認作業を効率的に行えるよう、各テストコードに伴った実行画面（ブラウザ画面）の内容を正常系テストの場合と比較し、差分を自動的に抽出・グルーピングすることで、テストの確認作業の効率化を図る。

2 関連技術

本節では本論文の提案内容に関連する技術を簡単に紹介する。

2.1 Selenium IDE と Selenium WebDriver

Selenium は Web ブラウザの自動操作を可能にするテストツールであり、主に Selenium IDE [2] と Selenium WebDriver [8] の 2 つの技術から構成されている。Selenium IDE は、ブラウザ上で操作を記録・再生する GUI ベースのツールであり、プログラミングの知識がなくてもテストスクリプトを作成できる点が特徴である。さらには、記録された操作内容の編集や、ステップ実行によるデバッグも可能であるため、手軽にテストの構築・確認が行える。Selenium WebDriver は、プログラムから Web ブラウザを制御するための API を提供しており、テストスクリプトをソースコードの形式で記述することで、より柔軟かつ高精度なテストの自動化が可能となる。本論文では、これらの技術を組み合わせて活用し、テスト支援ツールを構築している。

2.2 ページオブジェクト

本論文では、異常系テストの自動生成を容易にするため、ページオブジェクトモデルの概念を参考にしてプログラムの構造変換を行う [4]。ページオブジェクトモデルでは、Web ページごとの要素操作をオブ

表 1 ミューテーションの例

ミューテーション名	概要
命令の削除	あるソースコード行または命令を削除する
命令のローテーション	一連のコード行や計算式の順序をローテーションする
演算子の置換	演算子を別のものに置き換える
条件の否定	条件式を否定形に書き換える

ジェクトクラスとして定義し、各要素の識別情報（ID 等）をロケータとして一元的に管理することで、テストコードの可読性および保守性を高めることができる [7]。本論文では、各ページに対応する操作を関数単位で記述し、ロケータを活用する形式を採用している。これにより、ブラウザ操作のロジックと DOM 要素の特定処理を分離し、テストケースの生成と実行を効率化する。

2.3 ミューテーション技術

ソフトウェアテストの分野では、被テストプログラムの一部を敢えて書き換えた変異プログラムを作り出し、既存のテストスクリプトがそのような変異を誤りとして適切に検出できるか確認する“ミューテーションテスト”という手法が広く知られている。このような変異プログラムは“ミュータント”と呼ばれ、ミュータントを生成する際にプログラムの一部を敢えて書き換える操作のことを“ミューテーション”と呼ぶ。表 1 に代表的なミューテーションの例を示す [1]。本論文では、このミューテーション技術をテストスクリプトに適用することで、人手での設計が煩雑なテストスクリプトを正常系テストスクリプトから自動的に生成する手法を提案する。

2.4 差分抽出

変異テストスクリプトを自動生成・自動実行した後、その実行結果を確認しなければならない。現状では、この確認を完全に自動化するのは（“何をもちてテストに成功といえるのか”を個別の案件に対して自動的に定義することが）難しい。そのため、実行結果の確認は人手に依存せざるを得ないのが現状であるが、自動生成・実行される変異テストプログラムの件数が多くなると、すべてを手作業で確認することは

現実的ではない。そこで、そこで本論文では、変異テストプログラムの動作結果を効率的に分析できるよう、正常系テストスクリプトの動作結果との“構造の差分”と“内容の差分”を抽出する。構造差分は実行時に得られるHTMLのDOM構造の違いに着目したものである。一方、内容差分は、HTMLに構造の差分が見られない（DOM構造に違いはない）場合にのみ注目する差分であり、画面の表示内容（スクリーンショット画像）の違いを示すものである。

まず、構造差分の抽出においてはHTML文書のDOM構造に着目し、そこでのツリー編集距離[5]を用いる。ツリー編集距離の計算には、ノードの挿入・削除・変更といった基本的な編集操作を考慮し、動的計画法を用いて最適な変換コストを算出する。これにより、元となる正常系テストスクリプトと生成した各変異テストスクリプトの間におけるステートメント単位での構造的差異を定量的に測定できる。

次に、構造の差分が見られなかった場合には、Webページのスクリーンショット画像を対象に内容的な差分の抽出を行う。内容差分の抽出には、Pythonのscikit-imageライブラリを用いる。画像比較には構造的類似度指標（SSIM: Structural Similarity Index）を用い[9][12]、単なるピクセル差分だけでなく、構造的な視覚差にも対応する。SSIMにより得られた差分マップは、輝度値に基づいて二値化され、モルフォロジー処理および輪郭抽出によって差分領域が矩形として検出される。これにより、元となる正常系テストスクリプトと生成した各変異テストスクリプトの間におけるステートメント単位での内容的差異を視覚的に理解できる。

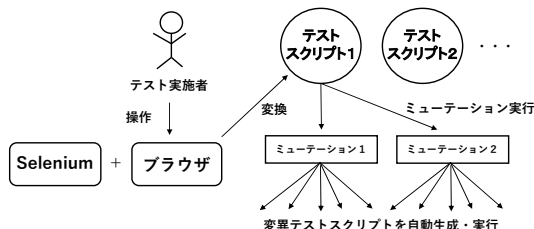


図1 変異テストスクリプトの自動生成までの流れ

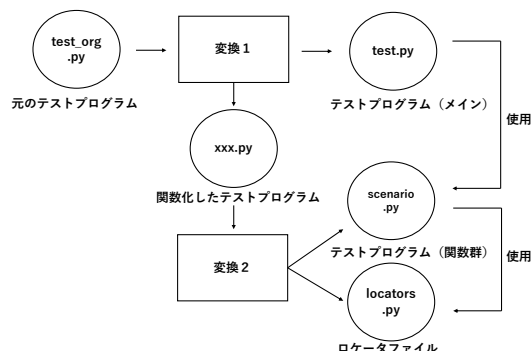


図2 プログラム変換の流れ

3 提案手法

本節では、変異テストスクリプトの自動生成および実行、並びに画面差分のグルーピングを以下の手順で行うことを提案する。

1. **正常系テストの記録**: Webアプリケーションに対して人手で正常系テストを実施し、その操作内容をSelenium IDEにより記録する。そして、記録したテスト内容をSelenium Webdriverを使ったPythonプログラムとして出力する(図1)。
2. **プログラムの変換**: 出力されたPythonプログラムをWeb操作の処理に関するプログラムとWeb要素の位置情報(ロケータ)に関するプログラムに分割して整理する。図2にテストスクリプト変換の流れを示す。変換は以下の2段階で行う。まず、記録したテストスクリプト(“test_org.py”)から関数化したテストプログラム(“xxx.py”)を作成し、“test.py”から関数を呼び出せるようにする。次に、“xxx.py”をロケータ定義部(“locators.py”)と操作手順部(“scenario.py”)に分割する。この変換により、ミュータント生成対象は“scenario.py”のみを対象に行えばよく、テストターは“test.py”を“pytest”で実行するだけでテストを自動化できる。
3. **ミュータントの生成**: Web操作に関するプログラムに対し、ミューテーション操作を施して変異プログラム、即ち、操作ミス等を模したテスト

ケースを自動生成する。採用するミュートーション操作は、ユーザーによる操作ミスの傾向を明らかにするために、システムデザインに関する書籍[6][10]を参考に調査・選定した。ここでは“テストコードの削除”，“テストコードの置き換え”，及び“ロケータの置き換え”という3種類のミュートーションを採用する。

- **テストコードの削除**：Web 操作に関する命令の1つ（1文）を元のプログラムから削除する。これにより、1つのWeb操作を“やり忘れた”というテストケースを生成できる（例：あるチェックボックスのチェックをやり忘れる）。
- **テストコードの置き換え**：1つのWebページに対して行うWeb操作命令について、その中の命令の順序をローテーションさせる。テストコードの置き換えという観点でいえば、任意の行と行を置き換える方法も考えられるが、組合せ数が大きくなりやすいという理由からローテーションのみを採用している。これにより、一連のWeb操作の“操作順序を間違えた”というテストケースを生成できる（例：あるフォームに入力する前に送信ボタンをクリックしてしまう）。

- **ロケータの置き換え**：本手法では、ミュートーションに先立って行うプログラム変換によって、Web操作（動作）そのものを表すコードとその対象となるWeb要素の位置情報（ロケータ）が分離されている。そして、ロケータを別のものに置き換えることで別のWeb要素に対して操作を実行するようプログラムを変異させる。この場合もテストコードの置き換えと同様にロケータに該当する部分だけをローテーションさせる。これにより、本来とは“別の要素に対して操作を行ってしまう”というテストケースを生成できる（例：本来とは異なるボタンをクリックしてしまう）。

また、ミュートーション操作で変異プログラムを生成する際に、同時に、後の差分抽出のためのHTMLとスクリーンショットを取得するコードを変異プログラムに追加する。

4. **テストケースの自動実行**：上述の手順で生成されたテストケースは、pytestコマンドを用いることで自動実行が可能である。それにより、Webブラウザが自動的に起動し、スクリプト内で定義された一連の操作が順次自動実行される。
5. **実行画面の差分抽出**：

自動生成された変異テストケースでは、その元となった正常系テストケースとはいずれかが異なる動作（主として操作ミス）が行われることになるため、Webブラウザでテストコードの実行中に表示される実行画面の内容には何らかの差異が見られることが期待される。この差分を抽出する流れを図3に示す。生成した変異プログラムを自動実行すると、各テストスクリプト内のステートメント単位でHTMLとスクリーンショット（SS）が自動取得される。これら各変異プログラムで取得したHTMLとスクリーンショットを、元の正常系テストを実行した際に生成されるものと比較する。そして、前述した構造差分を抽出し、構造差分がないと判定されたテストケースには内容差分を抽出する。このように、正常系

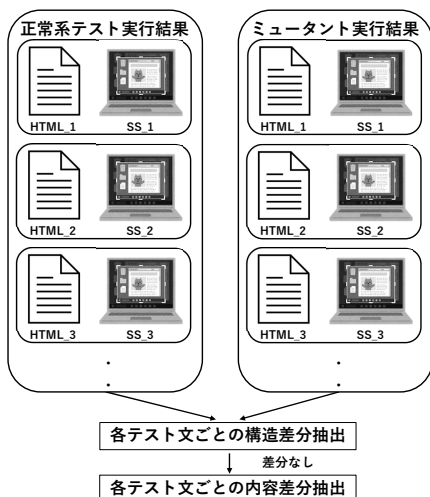


図3 テスト実行結果に対する差分抽出の流れ

の場合との差分を自動抽出して、差分に応じてグルーピングを行い、より効率的に想定外の結果を見つけやすくする。

4 事例研究

本節では提案手法の有用性について検討するために実施した事例研究について報告する。

4.1 目的

本論文では、Web アプリケーションに対する異常系テストの自動支援を目指している。ここでは次の2つの研究課題 (RQ) を掲げ、事例研究を通じてこれらに回答することを目的とする。

RQ1: 提案手法では変異テストプログラムをどの程度効率的に生成できるのか？

RQ2: 提案手法での差分に基づいた分類は動作確認として有効か？

RQ1 は、テスト工数の削減という観点から提案手法の有用性に関する問いである。本手法では、テスターが手作業で行った正常系テストをもとに、変異テストを Python プログラムとして自動生成する。もし生成される変異テストの数が少なく、手作業と比較して工数削減の効果が低ければ、本手法を新たに導入する価値は低いと考えられる。

RQ2 は、差分に基づいたグルーピングによって、生成した変異テストプログラムの実行結果を効率的かつ効果的に確認できるかという問いである。本手法で自動生成・実行した変異テストプログラムの実行結果を手で確認するのは現実的ではない。そこで、本手法では、変異テストプログラムごとに正常系と比較した実行画面の構造的・内容的な差分を自動的に抽出し、それらの類似度に基づいてグループ化することで、確認対象を視覚的に整理・分類することを可能にしている。本 RQ では、こうした差分グルーピング手法が実際のテスト確認作業の効率化や不具合の早期発見に寄与するかを検証する。

4.2 対象

本研究では、RQ1 と RQ2 それぞれについて検討するため、以下に示す2つの Web アプリケーション

を研究対象とした。

(a) 愛媛大学修学支援システム

RQ1, RQ2 について検討するため、愛媛大学修学支援システムを事例研究の題材とする。修学支援システムは、学生の履修や大学生活に関するさまざまな支援を提供する Web アプリケーションであり、履修に関する手続きや成績等の確認、講義に関する情報の検索といったさまざまな機能が提供されている。本研究では、テストの再現性を担保するため、同システムから何らかの手続きを行うようなテストや実行のタイミングによって結果が変化するようなテストは避けることとし、“シラバス検索” 機能に着目したテストシナリオについて検討を行う。

(b) 就活面談予約フォーム

RQ1, RQ2 について検討するため、簡易的な“就活面談予約フォーム”を研究対象の Web アプリケーションとして用意した (図 5)。

この就活面談予約システムには3つの画面があり、それらは (a) 予約画面、(b) 予約確認画面、(c) 予約完了画面である。まず、利用者が同システムへ Web ブラウザでアクセスすると予約画面 (図 5 (a)) が表示される。その画面では、会議室を予約するために表 2 に示す 13 項目すべてを記入ないし選択する。そして、“次へ” ボタンをクリックすると、予約確認画面 (図 5 (b)) が表示され、入力内容の確認を求められる。その後、“予約を確定する” ボタンをクリックすると、予約完了画面 (図 5 (c)) が表示される。その際、



図 4 愛媛大学修学支援システム

就活面談予約フォーム

氏名:

性別:
 男性 女性

メールアドレス:

大学名:

学部 (研究科):

学科 (専攻):

理系・文系:
 理系 文系

学年:

企業選びの軸:
 給与 事業内容 職場環境 福利厚生 キャリア成長 勤務地

予約日:

面談時間:
 開始時間を選択
 ~
 終了時間を選択

目的:

(a) 予約画面

予約内容確認

氏名: 山下 智也
 性別: 男性
 メールアドレス: abc@gmail.com
 大学名: 愛媛大学
 学部 (研究科): 理工学研究科
 学科 (専攻): 理工学専攻
 理系・文系: 理系
 学年: 大学院1年
 企業選びの軸: 給与 福利厚生 勤務地
 予約日: 2025-01-24
 面談時間: 14:00 ~ 15:00
 目的: 面接対策がしたいです

(b) 予約確認画面

予約完了

予約が完了しました!

(c) 予約完了画面

図 5 就活面談予約フォーム

表 2 入力内容

記入項目	入力形式
氏名	テキストボックス
性別	ラジオボタン
メールアドレス	テキストボックス
大学名	テキストボックス
学部 (研究科)	テキストボックス
学科 (専攻)	テキストボックス
理系・文系	ラジオボタン
学年	プルダウン
企業選びの軸	チェックボックス
予約日	プルダウン
開始時間	プルダウン
終了時間	プルダウン
目的	テキストボックス

内部では予約内容が担当者へ通知され、後日担当者から予約の可否が申請者へメールで通知されるという仕様になっている。

なお、このアプリケーションにはあえて次の 2 つの不具合を入れてある。

(a) メールアドレスが未記入でも送信できてしまう。

予約の可否を知らせる上でメールアドレスの入力は必須のはずであるが、同システムではこれが未記入 (空欄のまま) でもエラーとならず確認画面に遷移できてしまい、予約確定ボタンをクリックできてしまう。

(b) 予約時間に矛盾があっても送信できてしまう。面談を予約するにあたって、面談開始時間が面談終了時間と同じかそれ以降という予約は矛盾しているが、そのような矛盾した時刻を記入していてもエラーとならず確認画面に遷移できてしまい、予約確定ボタンをクリックできてしまう。

4.3 手順

本事例研究では以下の手順で変異テストスクリプトの生成と実行、並びに評価を行う。

- (1) 正常系テストの実行: 2 つの Web アプリケーションに対し、図 6, 7, 8 に示すシナリオに従って手作業で正常系テストを行う。ただし、修学

シラバス検索機能を選択後、検索項目の

- “開講年度” 欄から “2021 年度”，
- “開講学部” 欄から “工学部”，
- “開講学科” 欄から “工”，
- “開講コース” 欄から “応用情報工学コース”，
- “開講学期” 欄から “前期”

をそれぞれ選択して “検索” ボタンをクリックする。そして、検索結果に “知的グループワーク演習” の 1 科目が表示されることを確認する。

図 6 修学支援システム正常系テストシナリオ 1

シラバス検索機能を選択後、検索項目の

- “開講年度” 欄から “2022 年度”，
- “開講学部” 欄から “工学部”，
- “開講学科” 欄から “工”，
- “授業科目区分” 欄から “専門教育科目”，
- “対象年次” 欄から “3 年生”，
- “開講学期” 欄から “前期”，
- “開講学部・学科” 欄から “工学部”，
- “学年レベル” 欄から “学士 3 年生”，
- “開講曜日” 欄から “火曜日”，
- “開講時限” 欄から “4 時限”，
- “時間割番号” 欄から “15315”，
- “科目区分” 欄から “専門応用科目”，
- “分野大分類” 欄から “工学”

をそれぞれ選択して “検索” ボタンをクリックする。そして、検索結果に “データベース” の 1 科目が表示されることを確認する。

図 7 修学支援システム正常系テストシナリオ 2

面談予約フォームに表 2 の項目を正しい形式でそれぞれ入力・選択する。続いて、次の “予約確認画面” で “予約を確定する” ボタンをクリックして、 “予約完了画面” が出力されることを確認する。

図 8 就活面談予約フォーム正常系テストシナリオ

支援システムのテストシナリオにおいては、最初にユーザ ID “m807051u” で同システムにログインし、メニューから “シラバス検索” を選択するところから始める。テストの内容はすべて Selenium IDE で記録する。

- (2) 正常系テストプログラムの生成：手順 (1) で記録されたテスト内容を Python プログラムとして出力する。
- (3) 正常系テストプログラムの変換とそれに基づいた変異テストプログラムの自動生成：手順 (2) で生成されたテストプログラムに対して提案手法

表 3 各シナリオで生成されたミュータント数

対象	シナリオ	ミューテーション			合計
		削除	置換	置換	
修学支援システム	1	18	27	14	59
就活面談予約フォーム	2	34	57	30	121

(支援ツール) による変換並びにミューテーション操作を施し、変異テストプログラム (ミュータント) を自動生成する。

- (4) 変異テストプログラムの自動実行：手順 (3) で自動生成された変異テストプログラムをそれぞれ pytest コマンドを使って自動実行する。その際、ステートメントごとに実行画面のスクリーンショットと HTML コードも自動的に記録する。
- (5) 不具合の修正と回帰テストの実施：手順 (4) で得られた各変異テストプログラムの実行画面の HTML、スクリーンショットと元の正常系テストでの間の差分を抽出し、グルーピングを行う。差分グルーピングの結果から、対象の Web アプリケーションに想定外の結果が発生していないかを確認し、さらに就活面談予約フォームに混入している不具合を検出できるかを確認する。

不具合を検出できていた場合は同システムに対する改修を行い、不具合修正後のシステムに対してすべてのテストプログラムを使って回帰テストを行う。その際、不具合が適切に修正されていることを確認できるか確かめる。

4.4 結果

前述した手順を通じて得られた結果を以下に示す。

4.4.1 変異テストの自動生成と自動実行

上述した各シナリオに対してプログラム変換並びにミュータント自動生成を行った結果、表 3 に示す数のミュータント (変異テストプログラム) が得られた。

次に、自動生成されたミュータントを pytest コマンドで自動実行させ、その実行に要した時間を測定した。また、これらの変異テストを仮に手作業で実行した場合に必要な工数 (テスターが 1 人で実施した場合の所要時間と等価) についても概算した。

表 4 実行時間の比較結果

対象	シナリオ	自動実行 時間 (秒)	手動実行 時間 (秒)
修学支援 システム	1	771	1,302
	2	1,555	4,505
就活面談 予約フォーム	1	993	4,781

概算にあたっては、以下のような手順を採用した。まず、各ミュートーション操作（テストコードの削除・置き換え・ロケータの置き換え）の種類ごとに、ランダムに3個のミュータントをサンプル抽出した。その後、筆頭筆者が Web ブラウザを手作業で操作し、1つの変異テストケースを最後まで実行するのにかかる時間の平均を求めた。そして、各ミュートーション操作ごとの平均実行時間に、その操作によって生成されたミュータントの個数を乗じ、3種類について合計することで手作業で全ミュータントを実行した場合に要する総テスト時間の概算値を算出した。提案手法を用いて自動実行するのに要した時間（実時間）と手作業によるテスト時間（概算値）を表4に示す。ただし、提案手法での実行時間には、ミュートーションの元となる正常系テストスクリプトの手動実行に要した時間も含まれる点に注意されたい。

なお、この概算値は、休憩を一切取らず、操作ミス（入力ミスや選択ミスなど）も一切発生しないという理想的な条件を前提として算出されたものである点に注意されたい。したがって、実際の作業においては、これ以上の時間が必要になると思われる。

4.4.2 差分グルーピング

2つの Web アプリケーションを対象として各シナリオ（図6, 7, 8）から生成された変異プログラム（表3）を自動実行し、その結果得られた各変異プログラムの HTML およびスクリーンショットについて、正常系テストプログラムの実行結果と比較し、各実行画面との間に存在する差分をそれぞれ抽出した。その上で、差分に基づいて変異プログラムをグルーピングした結果、表5, 6に示す3種類のパターンが確認された。

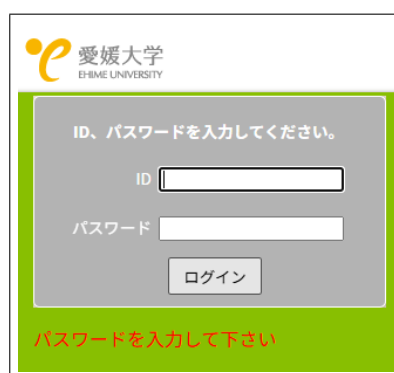
1つ目は構造差分が検出された変異プログラムのグループである。これは、ミュートーションの影響によ

表 5 修学支援システム差分グルーピング結果

シナリオ	構造差分あり	構造差分なし	
		内容差分あり	内容差分なし
1	48	10	1
2	85	35	1

表 6 就活面談予約フォーム差分グルーピング結果

シナリオ	構造差分あり	構造差分なし	
		内容差分あり	内容差分なし
1	72	15	0



(a) 愛媛大学修学支援システム



(b) 就活面談予約フォーム

図 9 差分グルーピング例（構造差分あり）

り、正常系テストスクリプトと比較して実行画面の HTML 構造に変化が生じた変異プログラムを分類したものであり、修学支援システムのテストシナリオ1では全59件中48件、テストシナリオ2では全121件中85件、就活面談予約フォームのテストシナリオ1では全87件中72件がこれに該当した。例

開講年度	2025年度	開講学期	工学部	開講学科	工
履修コース	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修シラバス	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修担当	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修担当	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修担当	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修担当	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修担当	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目
履修担当	工学部1科目	工学部1科目	工学部1科目	工学部1科目	工学部1科目

(a) 愛媛大学修学支援システム

予約内容確認

氏名: 山下 智也
 性別: 男性
 メールアドレス: abc@gmail.com
 大学名: 愛媛大学
 学部 (研究科): 理工学研究科
 学科 (専攻): 理工学専攻
 理系・文系: 理系
 学年: 大学1年生
 企業選びの軸: 福利厚生 勤務地
 予約日: 2025-01-24
 面接時間: 14:00 ~ 15:00
 目的: 面接対策がしたいです

(b) 就活面談予約フォーム

図 10 差分グルーピング例 (構造差分なし・内容差分あり)

例えば、図 9 (a) のように、修学支援システムではパスワードが未入力のままログインを試みた結果、エラーメッセージが表示されるケースや、図 9 (b) のように、就活面談予約フォームでは大学名を未入力のまま予約を試みた結果、エラーメッセージが表示されるケースがこれに含まれる。いずれも正常系とは異なる動作結果を示しており、そのまま進行しないことが期待される異常系テストとして問題ない事例であったといえる。

2 つ目は、構造差分は見られなかったものの、内容差分が検出された変異プログラムのグループである。これは、ミューテーションの影響により、HTML の構造自体には変化がなかったものの、画面の見え方や表示内容に変化が生じたミュータントを分類したものであり、修学支援システムのテストシナリオ 1 では全 59 件中 10 件、テストシナリオ 2 では全 121 件中 35 件、就活面談予約フォームのテストシナリオ 1 では全 87 件中 15 件がこれに該当した。例えば、図 10 (a) のように、修学支援システムでは開講学期

が未入力であったために検索結果画面で内容差分が生じたケースや、図 10 (b) のように、就活面談予約フォームでは企業選びの軸のチェックボックスでクリックしていない項目があったために、予約内容確認画面で内容差分が生じたケースがこれに含まれる。

3 つ目は、構造差分および内容差分のいずれも検出されなかった変異プログラムのグループである。これは、修学支援システムのテストシナリオでそれぞれ 1 件ずつ確認された。

4.4.3 不具合の検出と修正、並びに回帰テスト

4.3 節の手順 (5) で説明したように、自動生成された変異プログラムを用いて、就活面談予約フォームに意図的に混入させた不具合を検出可能かどうかを検証した。変異プログラムは操作ミスを模倣したテストプログラムであるため、それを実行したにもかかわらず予約申請が正常に完了してしまう場合には、不具合の可能性があると判断できる。差分抽出の結果、混入させた不具合のうち 1 件が、内容差分が生じたミュータント群 (図 10) の中に含まれていた。具体的には、図 11(a) に示すように、ミューテーションの影響でメールアドレス欄が未記入であるにもかかわらず、エラーが発生せず確認画面へと遷移し、そのまま申請ボタンをクリックできてしまっていた。

この不具合に対して Web アプリケーションの改修を行った後、すべてのテストを改めて再実行した。その結果、前述の不具合については、図 11 (b) に示すように、エラーとして正しく処理されるようになった。一方で、もう 1 つの不具合“予約時間に矛盾があっても送信できてしまう”という問題については、今回の変異プログラムでは検出には至らなかった。

4.5 考察

以下では実験結果に対する考察を述べ、各 RQ に回答する。

4.5.1 RQ1 に関する考察

本事例研究では 2 種類の Web アプリケーションとそれに対する正常系テストスクリプト (テストプログラム) をそれぞれ用意し、提案手法によって変異テストプログラムの自動生成と自動実行を行った。その結果、表 3 に示したように 3 種類のミューテーション

予約内容確認

氏名: 山下 賢也
 性別: 男性
 メールアドレス: [REDACTED]
 大学名: 愛媛大学
 学部 (研究科): 理工学研究科
 学科 (専攻): 理工学専攻
 理系・文系: 理系
 学年: 大学院1年
 企業選びの軸: 給与, 福利厚生, 勤務地
 予約日: 2025-01-24
 面接時間: 14:00 ~ 15:00
 備考: 面接対策がしたいです

予約を確定する
戻る

予約完了

予約が完了しました！

新しい予約をする

(a) 不具合が見つかった例

例: example@gmail.com

大学名:

学部 (研究科):

学科 (専攻):

理系・文系: 理系 文系

学年:

企業選びの軸: 給与 専攻内容 職場環境 福利厚生 キャリア成長 勤務地

予約日:

面接時間: ~

目的:

(b) 不具合修正後の実行結果

図 11 不具合の確認と修正

ン操作に基づいて各ミュータントを修学支援システムのシナリオ 1 に対して 59 個, シナリオ 2 に対して 121 個, 就活面談予約フォームのシナリオ 1 に対して 87 個の変異テストプログラムをそれぞれ自動生成できた。自動生成されたすべての変異テストプログラムが当該アプリケーションのためのテストケースとして等しく有用であるか否かまでは断言できないが, 少なくとも何らかの例外的な使い方や操作ミスに対応したテストケースにはなっているわけであり, それらを実行することで当該アプリケーションで暴走等の想定外の動きが起こらないことの確認には一定の価

値があると思われる。

また, 表 4 から分かるようにテストの実行のみならず, 変換及び生成にかかる時間全体も考慮しても, 手作業で行う場合に比べて提案手法による自動化で以下のようにテスト時間が削減される:

- 修学支援システムシナリオ 1:
1302 秒 → 771 秒 (531 秒減; 約 41%減)
- 修学支援システムシナリオ 2:
4505 秒 → 1555 秒 (2950 秒減; 約 66%減)
- 就活面談予約フォームシナリオ 1:
4781 秒 → 993 秒 (3788 秒減; 約 79%減)

シナリオが複雑になるにつれて自動実行と手動実行の差は顕著になり, たとえば修学支援システムのシナリオ 2 では約 50 分, 就活面談予約フォームのシナリオ 1 では約 60 分の工数削減につながった。繰り返しになるが, ここで示した手作業によるテスト時間は, 休憩を一切取らず, 操作ミスもないという理想的な条件下での概算値である。現実にはさらなる時間を要する可能性が高く, 実際の差はより大きくなると考えられる。自動実行が手作業より高速であることは当然とも言えるが, 現実の Web アプリケーションには多様かつ複雑な動作シナリオが多数存在するため, 本手法による効率化の効果は今後ますます大きくなると期待される。

以上より, RQ1 (提案手法では変異テストプログラムをどの程度効率的に生成できるのか?) には次の通り回答する: 提案手法は 1 個の正常系テストケースを手動で用意するだけで, 59-121 個もの操作ミス事例を含んだテストプログラムを自動生成でき, なおかつその生成と実行に要する時間も手作業の場合に比べて十分に短い時間で完了できることを確認できた。

4.5.2 RQ2 に関する考察

本研究では, 変異テストの実行結果に対する動作確認の効率化を図るため, 正常系との比較に基づく差分抽出と変異テストプログラムのグルーピングを行った。具体的には, テストスクリプト内の各ステートメント実行時に生成される HTML の構造 (DOM) を比較して構造差分を, スクリーンショット画像を比較して内容差分をそれぞれ抽出し, その差分の有無に基づいて変異テストスクリプトを分類・整理した。その



図 12 内容差分確認例

結果、表 5 および表 6 に示すように、以下の 3 種類のグループが得られた：

- 構造差分あり
- 構造差分なし・内容差分あり
- 構造差分なし・内容差分なし

このうち、構造差分ありのグループについては、操作ミスの影響で本来の遷移や画面構成が変化したものであり、正常系とは異なる動作結果となっているため、その意味ではテスト成功の事例であるといえる。一方、構造差分がないグループは、変異テストプログラムであるにもかかわらず正常系と同じ動作結果となっている可能性があるため、内容差分の確認が必要である。構造は同一でも表示内容が異なる場合、表面的には正常に見えても期待通りに動作していない可能性がある。さらに、表示内容も同一の場合でも、操作ミスを模倣した変異テストプログラムと元の正常系テストプログラムの実行結果が一致するため、対象の Web アプリケーションに潜在的な不具合が含まれている可能性がある。したがって、このグループに属する変異テストプログラムについては、動作結果を優先的に把握すべきである。

実際に、表 6 で示したように、就活面談予約フォームで構造差分は見られなかったものの、内容差分が検出されたグループが見られ、そこには 15 個のミュートが属していた。そして、その中の 1 つが“メールアドレスが未記入であっても送信できてしまう”という不具合に対応するものとなっていた。つまり、87 個ある中からまずは当該グループに含まれる 15 個を優先的に確認することでより効率的に不具合を検出

できたといえる。

図 12 は、正常系テストプログラムと変異テストプログラムの実行結果をステートメント単位で比較し、検出された内容差分を可視化した例である。差分が生じた領域はハイライトされており、画面のどの部分に変化があったかが一目で確認できる。この例では、変異テストプログラムにおいて“メールアドレスが未記入でも送信できてしまう”という不具合が発生しており、差分表示によってその挙動が容易に特定できた。

また、図中の各スクリーンショットのファイル名には、テストスクリプト内のステートメント番号が付与されている。これにより、差分が発生したタイミングをコード上で即座に特定でき、不具合発生箇所の原因追跡や修正作業を効率化できる。図 12 で説明すると、1～6 文目までは実行結果が同一であったが 7 文目実行時に内容差分が発生し、8～28 文目も内容差分が発生し、29 文目で再び同一の結果となった。7 文目はメールアドレスを入力する文であるが、変異プログラムではこれを削除しており、このことから不具合の原因を特定できた。同様に、図 10 に示した変異テストプログラムの動作結果についても確認できた。このように、本手法は内容差分の確認とステートメント番号の対応付けにより、動作結果を迅速に把握することを可能にする。

なお、構造差分・内容差分のいずれも検出されなかった変異テストプログラムは修学支援システムの各テストシナリオで 1 件ずつ存在した。これは、ロケータの置き換えによるミュートーション操作において、ロケータが同じものを置き換えていたため、変異

テストでありながら実行結果が正常系と完全に一致してしまっていた。そのため、ミューテーションの設計や自動生成の精度にも注意を払う必要がある。

以上より、RQ2（提案手法での差分に基づいた分類は動作確認として有効か？）には次の通り回答する：構造差分と内容差分に基づくグルーピングは、想定外の動作の検出と確認作業の効率化において有効であることを確認できた、

5 まとめと今後の課題

本論文では、Web アプリケーションに対する異常系テストの自動化支援策として、Selenium によって記録されて出力される Python テストプログラムをミューテーション操作によって変異させることを提案した。具体的には、いったん正常系テストシナリオの下で手作業によって Web アプリケーションの正常系テストを行い、その内容を Selenium で記録する。そして、記録された内容を Python テストプログラムのかたちで出力させた後、それをページオブジェクトパターンに従ったプログラムに自動変換し、その上でミューテーション操作を施してさまざまな操作ミス等に対応した変異テストプログラムを自動生成するという手法である。さらには、その実行結果の確認を効率化するため、実行画面の HTML コードとスクリーンショットを 1 文ごとに自動収集して正常系の場合との差分を調べ、差分の大きさに応じてミュータントを自動分類（グルーピング）する手法も提案した。

2 つの Web アプリケーションを題材とした事例研究を通じて、提案手法は多くの変異テストプログラムを自動生成して自動実行できること、さらには上述した差分グルーピングによって効率的に結果の確認を行えることを確認できた。

今後の課題としては次の 2 点が挙げられる。1 つ目は OSS のアプリケーションへの適用である。本研究では大学内のシステムと自作のシステムを対象としたが、より実践的かつ汎用的な評価のためには、OSS の Web アプリケーションを対象に提案手法を適用し、有効性や課題を検証する必要がある。2 つ目はミューテーション操作の拡充である。今回の事例研究ではテストコードやロケータを置き換えるミューテーション

は適用範囲が限定的である。今後は、既存の人間の操作ミスを模倣するミューテーションをさらに拡充し、より多様な操作を対象とすることが求められる。

謝辞 本研究は JSPS 科研費 23K11382, 25K15059, 25K15062 の助成を受けたものです。

参考文献

- [1] Agrawal, H., DeMillo, R. A., Hathaway, B., Hsu, W., Krauser, E. W., Martin, R. J., Mathur, A. P., and Spafford, E.: “Design of mutant operators for the C programming language,” Technical report, Purdue University, March 1989.
- [2] Leotta, M., Molinari, A., and Ricca, F.: “Assessor: a PO-based WebDriver test suites generator from Selenium IDE recordings,” Proc. 2022 IEEE Conf. Softw. Testing, V. & V., pp. 389–399, April 2022.
- [3] Leotta, M., García, B., Ricca, F., and Whitehead, J.: “Challenges of end-to-end testing with Selenium WebDriver and how to face them: A survey,” Proc. 2023 IEEE Conf. Softw. Testing, V. & V., pp. 339–350, April 2023.
- [4] Leotta, M., Clerissi, D., Ricca, F., and Spadaro, C.: “Improving test suites maintainability with the page object pattern: An industrial case study,” Proc. 6th Int’l Conf. Softw. Testing, V. & V. Workshops, pp. 108–113, March 2013.
- [5] Nierman, A., and Jagadish, H. V.: “Evaluating structural similarity in XML documents,” Proc. 5th Int’l Workshop Web & Databases, pp. 61–66, June 2002.
- [6] Norman, D.: 誰のためのデザイン?, 新曜社, 1990. 原書: The Design of Everyday Things.
- [7] Padhy, P., Nayak, P., and Vaidya, S.: “Web performance engineering - finding best data structure for automatically collected HTML locators,” Proc. 2nd Int’l Conf. Comp. Systems & Commun., pp. 1–6, March 2023.
- [8] Raghavendra, S.: *Python Testing with Selenium: Learn to Implement Different Testing Techniques Using the Selenium WebDriver*, Apress, New York, 2020.
- [9] scikit-image development team: “Structural similarity index,” https://scikit-image.org/docs/stable/auto_examples/transform/plot_ssim.html#id2, 2025.
- [10] Shariat, J., and Saucier, C. S.: 悲劇的なデザイン, オライリー・ジャパン, 2017. 原書: Tragic Design: The True Impact of Bad Design and How to Fix It.
- [11] Software Freedom Conservancy: “Selenium,” <https://www.selenium.dev/>, 2024.
- [12] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E.: “Image quality assessment: From error visibility to structural similarity,” *IEEE Trans. Image Process.*, Vol. 13, No. 4, pp. 600–612, 2004.