VEGA Systems
https://www.vega-net.co.jp/

自動テストの運用事例紹介

2024年12月4日

株式会社ベガシステム

■目次

1.どんなところで自動テストを運用していた？

2.自動テスト運用の目的

3. 1ヶ月のスケジュール

4.自動テストで使用していたツール

5.自動テスト実施フロー

6.自動テストのシナリオ

7.緊急時の自動テストフロー

8.まとめ

■どんなところで自動テストを運用していた？

■対象

大規模ウェブサイトのログインシステム関連機能全般

■規模感

エラー画面をのぞいて約150画面
これを主要なブラウザ（サイトのブラウザ使用率を見て判断）、PCとスマートフォンで実施

→毎月これを手動でチェックするのは規模的にも難しい、
しかし、ログインシステムなのでチェックしないと不具合発生時に致命的なため、
自動テストが導入された。

■自動テスト運用の目的

・デグレードバグのチェックを行うこと
新規のアップデート内容に関してはここでは手動でテストを行っていた。

dev環境では、全てのアップデートのマージ後、デグレバグが存在しないか確認するために
対象のブラウザにて、全ての経路（エラーを含む）を通り、
そのスクリーンショット・HTMLを取得し差分を確認する。

■1ヶ月のスケジュール

月頭から
1週目：新規アップデート内容のすり合わせ、差分予定の確認、自動テスト改善
2週目：前月のアップデート内容の適応、テスト
3週目：当月の自動テストの実施、確認
4週目：本番でのテスト、証跡格納、自動テスト改善

■自動テストで使用していたツール

VEGA Systems

・Seleniumベースで、カスタムしたツールを使用していた

→Excelで遷移先のタグ指定、waitなどを指定でき、シナリオ単位で管理していた。
→ノーコードで記述できるため（たまにJS書く必要はあったが…）
学習コストが低い

Pythonで書いた場合

```python
# Set up the Chrome driver
service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)
try:
    # Open the URL
    driver.get("https://www.vega-net.co.jp/")

    # Wait until the link is present and clickable
    wait = WebDriverWait(driver, 10)
    link = wait.until(EC.element_to_be_clickable((By.XPATH, '//a[@href="https://www.vega-net.co.jp/work/"]//span[text()="IT事業実績"]')))

    # Click the link
    link.click()

    # Take a screenshot
    screenshot_path = "screenshot.png"
    driver.save_screenshot(screenshot_path)
    print(f"Screenshot saved to {screenshot_path}")

    # Get the page source (HTML)
    html_source = driver.page_source
    html_path = "page_source.html"
    with open(html_path, "w", encoding="utf-8") as file:
        file.write(html_source)
    print(f"HTML source saved to {html_path}")

finally:
    # Close the driver after a delay to see the result
    driver.implicitly_wait(5)
    driver.quit()
```

■自動テスト実施フロー

自動テスト実行
- 全実行に10時間ほどかかるため、前日に実行

結果確認
- ツール側でシナリオ毎に成功/失敗判定が出る

実行ログチェック
- 想定と同じ遷移・挙動をしているかチェック

画像差分チェック
- 表示崩れ確認

明らかな差分がある場合は開発側に
連携→修正→再実行
で全ての自動テストが完了するまで行う。

HTML差分チェック
- 差分が（ランダムな値を除き）追加アップデートのもののみかどうかを確認

■自動テストのシナリオ

画面遷移図にて、全ての画面の移動を行えるように
・各画面の起点から終点までを1シナリオとして作成
→エラーバリデーションを含めて全て網羅することで網羅率100%で全ての画面の表示崩れが担保される
※この事例の場合、全ての挙動が担保できるわけではないので注意

緊急の修正時にも該当する画面を含めた自動テストを短時間で実行できるよう、
・画面ごとにある程度シナリオを分ける
・どの画面がどのシナリオに含まれているかの資料を作成しておく
などの対応なども行っていた。

■緊急時の自動テストフロー

緊急時なので全ての自動テストを回している時間はない
→該当画面の自動テストを抽出し、
　先にその部分のみ回す。

→完了後再度デグレ確認として全ての画面を回す。

緊急時にスムーズに対象の特定・対象自動テストの実行を行うために
シミュレーションをして実際の動きを練習・かかる時間の測定などを行った。
→そのために各シナリオでかかる時間を計測などもおこなった。

■まとめ

・システムが大規模になればなるほど、手動でのデグレチェックを行うのは難しくなる。
→時間を短縮するために、自動テストという手段を使用する。

・ただ自動テストを導入するだけでなく、自動テストが担保できる範囲を
しっかり考える必要がある。

・自動テストで品質を担保する場合、通常時の運用だけでなく、
緊急時も自動テストを回さないといけないため緊急時の運用についても考慮する必要がある。