製品開発の生産性向上に向けた GUI テスト自動化への取り組み

Automating GUI Testing for Improving Productivity of Product Development

今後，業務向けソフトウェア市場は，マイクロサービス時代に突入することが予想される。市場の求めるものを迅速に提供するために，(株)日立ソリューションズ東日本(HSE)は，主力製品である SynViz S2, AppSQUARE について，開発プロセスを従来のウォータフォールモデルから DevOps へ移行中である。DevOps の必要要件としてテスト自動化があげられる。HSE の製品開発において，D 層，F 層のテスト自動化は事例があるが，GUI テスト自動化の適用は進んでいなかった。GUI テスト自動化技術が確立されているにもかかわらず，適用が進まない原因を分析した結果，Page Object Design Pattern を採用したテストプログラムの標準化，テストプログラムのライブラリ化，開発者を対象とした勉強会の開催により，導入時の障壁を軽減できることがわかった。この技術を製品開発に適用する見通しが得られたため，SynViz S2 の次期バージョンでの本格適用をめざす。

松本　和芳　Matsumoto Kazuyoshi
門司　太郎　Monji Taro

1. はじめに

急速に変化する顧客ニーズに対応するために，業務向けソフトウェアの市場は，マイクロサービス時代に突入することが予想される。この潮流に対応してソフトウェアを迅速に提供するためには，ソフトウェア開発プロセスをDevOpsにシフトする必要がある。HSEも，主力製品であるSynViz S2，AppSQUAREについて，開発プロセスを従来のウォータフォールモデルからDevOpsに移行中である。

DevOpsでは，継続的な品質保証を行うために，リグレッションテストが重要である。このテストを手動で行うと同じ操作を何度も繰り返し行うため，見落としや勘違いが発生しやすい。確実かつ効率的にテストを行うためにはテストを自動化して，このような問題を発生しないようにすることが必須要件となる。

2. GUI テスト自動化による生産性向上

2.1 製品開発に対するテスト自動化の現状

製品開発案件で，D層，F層に対する単体テスト工程(UT工程)では，テスト自動化が進んでいる。実際HSEの開発案件でも，複数のプロジェクトで，Jenkins[1)]やMocha[4)]+Chai[5)]といったテストフレームワークをUT工程に適用した事例がある。

UT工程でのテストの大部分は，APIを呼び出して，それが期待どおりの結果を返却するかを確認することが目的であるため，テストプログラム(TP)も設計が容易である(図1)。

```
// テスト条件
let id = 'test';
let password = 'p@55w0rD';

// テストの実施
const result = login(id, password);

expect(result).to.true();
```

図1 TPの例(UTの場合)

しかしながら，GUIテストを行う組合せテスト工程(CT工程)以降やリグレッションテストでは，依然としてチェックリストを元に画面操作を行い，画面を目視で確認するテスト消化方法を継続している。

その一方で，ブラウザ操作を自動化するツールは，

OSS (Open Source Software)であるSelenium[2]をはじめとして数多くリリースされている。

そこで生産技術部では，製品開発のGUIテストにこれらのツールの適用が進まない原因を分析し対策を打つことで，GUIテストを自動化し，生産性を向上させることができないか検討した。

2.2 GUIテストの自動化に対する課題の要因分析

SynViz S2，AppSQUAREの開発元である開発本部では，製品開発のUT工程にテスト自動環境をすでに導入していた。同様に，Seleniumを適用しGUIテストの自動化も検討を進めていたが，プロジェクトへの適用にはいたっていなかった。

(1) Seleniumの機能評価

そこで生産技術部は，GUIテスト自動化ツールの機能不足が適用の妨げになっていないかを評価するために，開発本部が適用の検討を進めているSeleniumの評価を行った。Seleniumは，最も著名なGUIテスト自動化ツールのひとつであり，マウスやキーボードなど入力デバイスのエミュレーションによって,外部からブラウザを操作することができるツールである。評価は,開発本部が開発している製品のリグレッションテストのチェックリストに基づいて図2の評価項目で行った。

1 環境面
(a)マルチブラウザ/マルチプラットフォーム対応か
(b)テスト実行環境とブラウザ実行環境を分離できるか
(c)同一スクリプトで複数環境を動作させることができるか
2 機能面
(a)HTML構造解析ができるか
(b)ページに対して任意のイベントを発生させることができるか
(c)入力デバイスをエミュレートできるか
(d)表示待ち受けを実現できるか
(e)キャプチャを取得できるか
3 非機能面
(a)ブラウザ表示について極端に性能向上または性能劣化しないか

図2 Seleniumの主な評価項目

評価の結果，確認項目のおよそ9割が実現できるため，GUIテスト自動化ツールの機能不足が適用の妨げになっていないことがわかった。

(2)開発本部への聞き取り調査

次に生産技術部は，運用面で課題があるという仮説をたて，開発本部の製品開発メンバに対して，GUIテスト自動化が適用されない原因の聞き取り調査を行った。その結果，以下の4点の課題が浮き彫りになった。

[課題1] TPが保守性の低いコードになってしまう

Seleniumを直接利用してTPを作成した場合，テストの本質ではない「ブラウザの起動」や「コントロール(画面の構成要素)の検索」,「コントロールへの操作」などがTPの大部分を占め，テストの本質である「チェック条件」,「確認項目」が埋もれてしまい，TPの可読性が低下する。その結果，保守性の低いコードになってしまう。さらに，このような設計では，コントロールの追加やレイアウトが変更になった場合，「コントロールの検索」がTPの各所に点在するため，影響範囲が拡大し，バージョンを重ねるたびにTPの維持コストが増大することが予想される。

このことを，対象製品のログイン画面のTPを例に挙げて説明する。図3は，特定のURLを開き，テキストボックスやボタンを検索後，検索したコントロールを操作してページを遷移させるTP例である。テスト結果の確認として，タイトルが「ホームページ」になっていることの確認を行っている。

```
// ログイン画面の表示
await driver.get('http://192.168.0.1/login');
await driver.wait(isPageShown);
```
ブラウザで表示する。

```
// テスト条件
let id = 'test';
let password = 'p@55w0rD';
```
テスト条件

```
// 画面要素の検索
const txtID
    = await driver.findElement(
        By.xpath('//*[id="loginid"]'));
const txtPass
    = await driver.findElement(
        By.xpath('//*[id="password"]'));
const btnLogin
    = await driver.findElement(
        By.xpath('//*[id="login"]'));
```
コントロールの検索

```
// テスト
await txtID.sendKeys(id);
await txtPass.sendKeys(password);
await btnLogin.click();
```
コントロールの操作

```
// 確認
const title = await driver.getTitle();
```
タイトルの取得

```
// タイトルの確認
expect(title).to.eq('ホームページ');
```
値の確認

図3 Seleniumを直接使った場合のログイン画面のTP

このように，Seleniumを直接利用した場合，コントロールの検索やコントロールの操作がTPの大部分を占め，テストの本質が埋もれてしまう。さらにコントロールの追加やレイアウト変更があった場合，TPすべてについて，「コントロールの検索」の部分を変更する必要があり，影響範囲が広くなる。

[課題2] TP開発の効率が悪い

Seleniumには，コントロールの検索では，図3で示したxpath以外にも，様々な検索メソッドが準備されている(表1)。

表1 コントロールの検索方法

項番	検索メソッド名	概要
1	id	HTMLタグのID要素を完全一致で検索する。
2	name	HTMLタグのname要素を完全一致で検索する。
3	js	要素オブジェクトを返却するJavaScriptを記述する。
4	className	HTMLタグのclass要素を部分一致で検索する。
5	xpath	W3C標準xpathで要素を検索する。

新規開発案件で，プロジェクト計画時からGUIテスト自動化を導入することが決まっている場合，詳細設計またはコーディングルールによって，idやnameの命名規則

を設計しておくことで，すべてのコントロールに対して意図的にidやnameを付与できる。しかし，idやnameはHTMLの規約によって必須ではない属性であるため，HSEの製品開発ではidやnameが省略されていることが多かった。また，近年Webアプリケーションの開発で使われる機会が多いGUIコンポーネントは，idやnameを指定しなかった場合，自動で動的に独自のidを付与することもある。この場合，コントロールを一意に特定するために，ほかの検索メソッドを使ってコントロールを検索することになるが，画一的な手段がない。したがって，すでに画面がある程度完成している場合，対象画面に対して，動的に生成されるHTMLをひとつずつ解析して，対象コントロールごとの特定方法を，図2で示した検索メソッドを組合せて設計する必要がある。これがTP開発工数の大半を占めることになり，TP開発の生産性が上がらない要因となる。

[課題3] テストエビデンスを手動で取得できない。

従来のテスト方法でGUIテストをする場合，画面の何をどのように確認したのかというエビデンスとして，画面キャプチャを残す手法をとってきた。しかし，ツールの自動化では画面遷移が速く，キャプチャを撮ることができない。

[課題4] TP開発者の学習工数がかかりすぎる。

テスト担当者が新たなGUIテスト自動化ツールを使う際，そのツールの習熟に工数を要し，結果的に生産性が上がらない可能性がある。

2.3 生産技術部の施策

以上の分析を踏まえて生産技術部では，HSEの開発本部に対して，段階的にGUIのテスト自動化を適用する施策を実施した。

(1) Page Object Design Patternを採用した設計標準

[課題 1]に対して，Selenium は，Page Object Design Pattern と呼ばれる設計技法を提唱している [3]。Page Object Design Pattern では，TP が「Page Object」と「テストシナリオ」の 2 つのモジュールからなるように設計することが推奨されている(図 4)。ここで，テストシナリオは従来のチェックリストに相当する。すなわち，テストシナリオにはテスト条件と確認項目を記述することが推奨されている。こうすることで，テストシナリオの可読性を上げ，TP のメンテナンス性を向上させることができる。

生産技術部は，このPage Object Design Patternを十分理解していなくともTPが自然にPage Object Design Patternに従う形式になることを目的として，SeleniumのAPIを隠ぺいする基底クラスの集合をTPのテンプレートとして提供した。これらの基底クラス群には，画面表示の待ち受け，ボタン上のテキストによる検索，ラベル名によるテキストボックスの検索など，テストでよく用いられる機能を実装した。図5に，本基底クラス(図ではPageクラスと表記)を継承してTPを作成した場合のクラス図を示す。

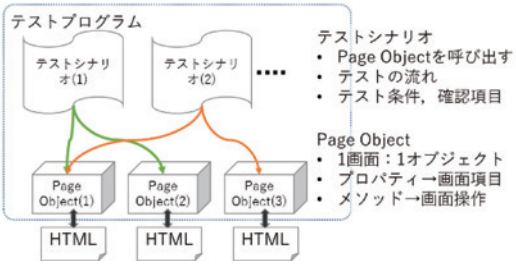

図4 Page Object Design Pattern概念図

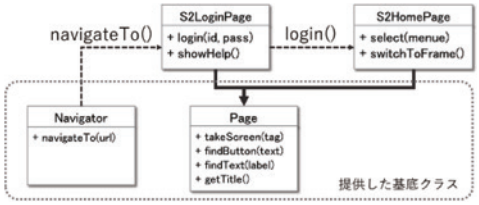

図 5 対象製品 TP のクラス図

このようにテスト担当者は，テスト対象となる画面に合わせて，「コントロールの検索」，「コントロールへの操作」を隠ぺいした Page Object(図 5 の S2LoginPage, S2HomePage)を提供された基底クラスを継承して実装する。Page Object を実装することで，これらを呼び出すテストシナリオの可読性が上がる。前節で示した，対象製品のログイン画面の TP は図 6 のようになり，可読性が向上したことがわかる。

図6 Page Object Design Patternの適用

こうして実装したPage Objectは，「テスト条件」や「確認項目」が異なるチェックリスト間で共有でき，TP開発効率向上につなげることができる。

(2)ライブラリ化

対象製品は，OSSのGUIコンポーネントを活用してい

る。これらのコンポーネントが出力するHTMLは，以下の特徴を持つ。

(a) HTMLの階層が非常に深い。

(b) 対象のコントロールがidやnameで一意に特定できない場合がある。

(c) ユーザの操作やサーバからの応答などの非同期処理で，HTMLの構造が動的に変化する。

[課題2]に示したとおり，このような特徴のHTMLでは，TP開発の生産性を上げることが難しい。そこで生産技術部は，対象製品の画面(HTML)を分析し，テスト対象になる可能性が高いコントロールを選び「コントロールの検索」と「コントロールへの操作」を隠ぺいするライブラリ群を提供した。また，ブラウザの開始，終了，URLへの移動，スクリーンショットの取得など，テストでよく使われる機能についてもライブラリ化し開発本部に提供した。

(3)勉強会の実施

開発本部は，所属メンバの技術レベルを維持/向上することを目的とした勉強会を定期的に開催している。生産技術部は本技法を習得してもらうために,その勉強会に参加した。勉強会は，TP 開発の実習を行うハンズオン形式とした。参加者は 8 から 10 名で，計 4 回実施した。各回のテーマは以下のとおりである(表 2)。

表2 勉強会で使用したテストシナリオ

回次	実装したテストシナリオ
第1回	ログイン，コピーライト/バージョン番号の確認，ヘルプの表示
第2回	コントロール（Button, TextBox, Grid, Form）の操作
第3回	全画面のキャプチャ取得
第4回	リグレッションテスト（要素の追加シナリオ）

これらのテーマは，製品開発で実際使用しているリグレッションテストのチェックリストからシナリオを選定した。ハンズオン形式にすることで，開発者の疑問にその場で答え，習熟に対する障壁をできるだけ低くすることに寄与できたと考える。また，ライブラリの設計不備(使いづらさ)をフィードバックすることで，利用者視点にたったライブラリやサンプルコードを提供することができた。

2.4 施策の評価

GUIテスト自動化に関するアンケートを，勉強会の参加者8名に対して実施したところ，回答者全員から技法に沿ったテストシナリオを設計し開発できる，という回答を得ることができた。そのうち2名からは，Page Objectの設計と開発を独力で実施することが可能という回答を得られた。この結果から，この2名をPage Objectの設計を担当する共通チームに割り当てることで，開発本部の中に本技法に沿ったTPを開発する体制を構築する見通しを得ることができた。

3. おわりに

この取り組みで得られたノウハウは，品質保証部にも共有する予定である。品質保証部は，早期に品質を安定させることを目的として，自身のテスト観点を製品開発の上流工程から取り込むことが望ましいと考えている。生産技術部は，品質保証部と連携して製品開発の生産性と品質の向上を図り，HSE の事業に貢献する考えである。

開発本部では，SynViz S2 の次バージョン開発から，本技法を本格適用する計画である。リグレッションテストから順次適用範囲を広げ，最終的には，CT 工程全体を網羅することを目指す。また生産技術部は，今回整備した GUI テスト設計技法を，他の HSE 製品開発に順次適用拡大するとともに，HSE の事業拡大に向けた製品のマイクロサービス対応や開発環境の DevOps へのシフトに寄与していく。

参考文献

1) Jenkins, https://jenkins.io/, Accessed 2019/9

2) Selenium – Web Browser Automation | SeleniumHQ, https://www.seleniumhq.org/, Accessed 2019/9.

3) Test Design Considerations | SeleniumHQ, https://www.seleniumhq.org/docs/06_test_design_considerations.jsp, Accessed 2019/9.

4) Mocha the fun simple, flexible JavaScript test framework | Mocha, https://mochajs.org/, Accessed 2019/9.

5) Chai, https://chaijs.com/, Accessed 2019/9.

松本　和芳　　2001 年入社
プロジェクトマネジメント本部
生産技術部
生産技術エンジニア
kazuyoshi.matsumoto.fd@hitachi-solutions.com

門司　太郎　　1994 年入社
開発本部
第一パッケージ開発部
製品アーキテクト
taro.monji.zc@hitachi-solutions.com