第57回 プログラミング･シンポジウム 2016.1.8-10

GUIテスト効率化のための継続的仕様書インテグレーション

上田 真史[1,a)]　岡崎 光隆[1,b)]

概要：GUI をもつアプリケーションソフトウェアをテストするとき、その手引きとして画面仕様書やワイヤーフレームと呼ばれる仕様書類が使われることがある。この仕様書は図および説明文からなるもので、たとえば画面の図があり、その各部を操作するとどのような効果があるのか等の仕様項目が文章で説明されている。ある程度熟練したテスターであれば、この仕様書をベースに、詳細な手順書なしにテストを実施できるので、テスト発注者の手間を大幅に削減できる。このテスト方法の精度や効率を上げるため、画面仕様書のチェックリストの作成と、分割した画面仕様書の統合を行う、継続的インテグレーションシステムを構築した。この論文では、その構築方法を述べ、実際に長期間運用した結果得られた効果や問題点について紹介する。

キーワード：テスト、継続的インテグレーション、画面仕様書、ワイヤーフレーム

1. はじめに

スマートフォンのアプリや web サービスなど、GUI をもつアプリケーションソフトウエア（以降「アプリ」という）を開発する際に、そのベースとなる資料として、画面仕様書またはワイヤーフレームと呼ばれるものがよく用いられる。プログラマー、グラフィックデザイナー、テスターなどがこの仕様書をもとに開発や検証の作業を行うが、この論文では主にテスターの利用を対象とする。

テストはアプリの品質を保つために重要なタスクである。その方法論は多岐にわたるが、本論文では画面仕様書をベースとして GUI をテスターが操作するテストをとりあげ、その精度向上や効率化のために構築した継続的仕様書インテグレーションシステムについて述べる。このシステムは実際に 18 ヶ月以上の運用実績があるので、運用によって実際に解決された問題や、解決しきれなかった問題について紹介する。

2. 画面仕様書

画面仕様書とは、図 1 のように、画面の見た目の図に、その各部への注釈を記述したものである。ワイヤーフレームとも呼ぶ。注釈には、その画面要素を操作したときに何が起こるかや、その画面要素の表示方法、出現条件などが含まれる。

アプリ開発の場においては、アプリの外からの見た目を定義するものとして、この仕様書が画面数ぶん用意される。内部モデルは別途定義される。プログラマはこの仕様書に沿ってプログラムを書き、グラフィックデザイナーはこの仕様書に沿って画面のアイコンなどを用意する。

[1] リプレックス株式会社
[a)] ueda@ripplex.com
[b)] okazaki@ripplex.com

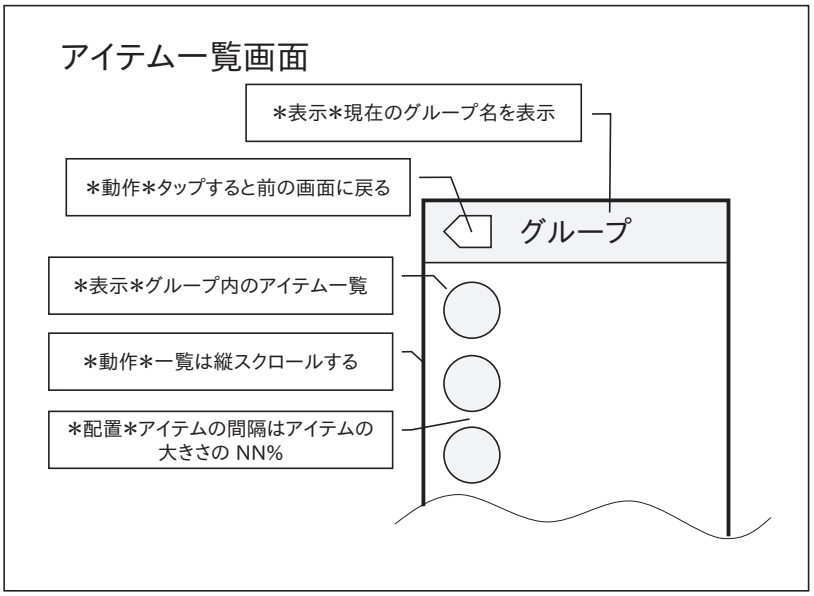

図 1 画面仕様書の例

3. 画面仕様書を用いたテスト

テスターがアプリをテストする場合にも、この画面仕様書を用いる方法がある。

アプリに限らず、プログラムの最も精密なテスト方法は、詳細なテストケースを構築してテストを行う方法である。これは、テストで何を行うかがステップ・バイ・ステップで厳密に指示されているものである。たとえば、画面のどのボタンを押すか、ということからはじまり、入力する文字列や数値までも指示に含まれる。テスターはこれをなぞるだけなのでテストは精密だが、まず手順書を構築しなければならないので、テスト発注者にとって非常にコストが高い。

一方、画面仕様書が用意できていれば、テスターはそれを読解してアプリをテストすることができる。たとえば、アプリで行う作業のゴールが決められていれば、ある程度の習熟は必要だが、画面仕様書をたどりながらテストができる。さらに、テスターの裁量に任せて操作のバリエーションをつけた上で反復してテストをすることで、カバー範囲を広げられる。テスターの主観が入るので、具体的な指示である手順書を使ったテストに比べ、操作にブレが生まれ、精度は下がるが、実際のテストとしてはそれで充分な場合が多い。

なにより、手順書を作成しなくてよいのは工数的にメリットが大きい。そこで、画面仕様書だけでテストをすることとし、このテストを補助する仕組みを作成することでテストの精度や効率をさらに高められないか、実際に補助システムを作成して確認することにした。

4. 効率化のポイント

画面仕様書をベースとしたテストを効率化するにあたって、以下のポイントをターゲットにした。

4.1 テスト漏れの対策

画面仕様書は、画面の図形に対して注釈がつけられるというスタイルで書かれるので、テスターが注意すべき点が仕様書上の様々な座標に散っている。散らばった点すべてに注意を向けるのはテスターにとって負担であり、相当の注意を払わないと、テストの漏れが発生する。後述する仕様書ファイルの分割を行うと、注意を払うべき座標軸が事実上もうひとつ増えることになるので、この傾向は強まる。テスト漏れ防止し、負担なく高精

度なテストを実施できる仕組みをつくりたい。

4.2 仕様書の作成の効率化

アプリの規模が大きい場合、画面仕様書の規模も大きくなる。この場合、複数人で担当範囲を分担し、並行して仕様書を作成したり更新したりできると効率が良い。これは単純にファイルを分割することで可能だが、同時に、テスト時に大量の画面仕様書ファイルを行ったり来たりしなければならないという問題が生まれる。この不便さは、テスト以外の作業に画面仕様書を使う場合にも同様である。複数人が並行作業できつつ、取り回しのよい画面仕様書ができるようにしたい。

5. 解決の方針

以上のターゲットについて、以下の 2 つの方法を組み合わせることで、解決を図ることにした。

- 画面仕様書の項目のチェックリストを作成する。画面仕様書に 2 次元的に散らばっている項目を 1 次元のリストにすることで、注意すべき点を単純化し、テストの際の取りこぼしを防ぐ。
- 分割した画面仕様書ファイルを連結する。複数の仕様作成者がそれぞれ別々の担当範囲のファイルを編集することとし、それらのファイルを連結して単一のファイルにすることで、取り回しをよくする。

これらの作業を自動で行うこととし、さらにバージョン管理システムと継続的インテグレーション（CI）システムを組み合わせ、画面仕様書 CI システムとすることとした。

一般的なソフトウエアの CI システムに対比すると、分割した画面仕様書がソースコード、結合してひとつになった仕様書とチェックリストが成果物である。ソフトウエアは計算機が実行するが、画面仕様書とチェックリストはテスターが実行する。

6. 仕様書 CI システムの構成

作成した画面仕様書 CI システムの構成を、図 2 に挙げる。以下、このシステムの各部の設計について述べる。

6.1 画面仕様書作成アプリの選定

ソースコードをコンパイルするにはその言語を規定しなければいけない。同様に、本システムでも対象の画面仕様書のファイル形式をまず決定した。画面仕様書は図とテキストの集合体であるので、ドロー系のアプリケーションソフトウエアであれば何でも使うことができる。本システムでは、PowerPoint[1] を採用した。

PowerPoint はプレゼンテーションスライド作成ソフトウエアであるが、同一レイアウトのスライドを複数枚まとめたファイルができるので、同じレイアウトの仕様書が画面ごとに必要な画面仕様書には都合がよい。プロプライエタリなソフトウエアであり、このこと自体は新たにシステムを構築するのにあまり望ましい選択肢ではない。しかし、普及の度合いが高い点と、オフラインでも作業できる点を重視して選択した。

普及の度合いが高い点は、特に外のチームと合同でプロジェクトを進める必要がある場合に、その担当者とデータのやりとりができるという点で重要である。オフラインで作業できる点は、作業者の PC が必ずしもオンラインでないことから、作業の効率のために必要であった。

6.2 処理スクリプト

各解決方法の自動化のために、主に以下の 2 つのスクリプトを作成した。ともに、PowerPoint ファイルを（間接的に）操作できる PowerShell[2] のスクリプトであり、PowerPoint のソフトウエア本体を呼び出して仕様書ファイルの解析および出力を行う。

6.2.1 チェックリスト生成スクリプト

チェックリスト生成スクリプトは、PowerPoint ファイルを読み、その中にある文字列をすべてスキャンする。正規表現パターンにマッチする文字列があったら、それを抜き出し、CSV（Comma Separated Value）として出力する。パターンは事前に規定しておき、仕様作成者が画面仕様書に項目を入力するときに、そのパターンに従って記述する。CSV の出力には、ソースファイル名やスライドのヘッダーなどのメタ情報をカラムとして含

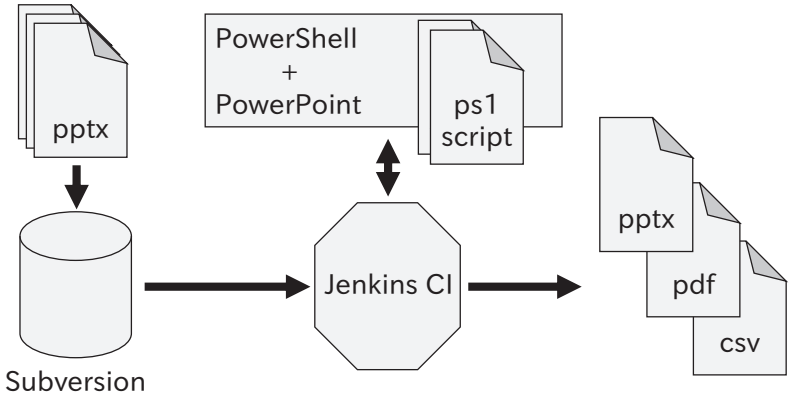

図 2 仕様書 CI システムの構成

めることもできる。

文字列のピックアップは PowerPoint の COM オブジェクト経由で行い、CSV 化は PowerShell の組み込みコマンドを使う。例として、図 1 の画面仕様書の例をこのスクリプトで処理すると、図 3 のような CSV データができる。

この例では、スライドのヘッダーを画面の名前とし、全角「＊」ふたつに囲まれた文字列を項目分類、それに続く文字列を説明文としている。

6.2.2 PowerPoint 連結スクリプト

PowerPoint 連結スクリプトは、1 つのディレクトリにある PowerPoint ファイルをすべて連結して、ひとつの PowerPoint ファイルにする。これは単純に、ファイルを辞書順でならべ、最初のファイルに 2 番目以降のファイルを追加していく。連結はファイル名順に行い、ファイル個数に制限はないので、画面 1 つごとにファイルを分割することも可能である。

出力は PowerPoint ファイルと PDF ファイルの 2 つとした。PDF ファイルは、共有相手が PowerPoint を読めない場合に備えてのバックアップ手段という位置づけで出力するようにした。

6.3 CI 環境の構築

以上のスクリプトを、Jenkins CI[3] で自動実行するシステムを構築した。

Jenkins は Subversion や Git などのバージョン管理リポジトリをポーリングし、更新があったらスクリプトを走らせる、ということができる。この仕組みを利用し、分割仕様書ファイルは Subversion[4] に置くこととした。ソフトウエアの CI 環境と同じく、仕様書記述者が仕様を更新し、Subversion リポジトリにコミットするだけで、統合した仕様書と仕様のチェックリストができる。

7. 運用の結果

この画面仕様書 CI システムを実際に、2013 年初から 2014 年秋まで、約 1 年半の間運用した。システムにより便利になったこと、また運用上扱いにくい点も明らかになったので、紹介する。

7.1 得られた効果

当初の目的通り、テスターは常に最新の画面仕様書とチェックリストを元にテストを行うことが可能になり、容易に漏れのないテストが可能になった。また、担当範囲ごとにファイルを分割して仕様書編集ができるので、仕様書作成の効率が上がった。

加えて、想定外のメリットとして、チェックリストの差分をとりやすいということがあった。これは、チェックリストが単純な CSV 形式であることに由来する。以前テストを行ったときのチェックリストと新たにテストを行うチェックリストの差分をとると、どこが更新されたのかがすぐにわかる。テストの際にその周辺を優先的にテストすることで、テストの省力化ができた。

```
アイテム一覧画面,表示,現在のグループ名を表示
アイテム一覧画面,表示,グループ内のアイテム一覧
アイテム一覧画面,動作,タップすると前の画面に戻る
アイテム一覧画面,動作,一覧は縦スクロールする
アイテム一覧画面,配置,アイテムの間隔はアイテムの大きさのNN%
```

図 3 チェックリスト出力例

さらに、仕様変更や機能追加にテスターが迅速に追随できるので、デザイナーやプログラマーの実装漏れも早期に発見し、開発の全体的なスピードアップができた。

7.2 扱いにくかった点

扱いにくかった点は、主に 2 点である。ひとつはアプリを拡張する際に画面仕様書を対応させることの困難、もうひとつは CI サーバーの管理の面倒である。

7.2.1 アプリ拡張への対応の困難

PowerPoint ファイルは、1 つのファイルの中のスライドはすべて同じ大きさで、かつそのレイアウトを定めるスライドマスターは 1 つしか持てない、という制約がある。

PowerPoint 連結スクリプトで複数の PowerPoint ファイルを連結すると、それらのファイルのうち最初のファイルのスライド寸法とスライドマスターが使われ、2 つめ以降のファイルの中身は自動的に、かつ強制的に、寸法とマスターが変更されて連結される。

これが行われると、大抵の場合、スライドのレイアウトが崩れ、めちゃくちゃくになる。こうなるとテスト業務に支障を来すため、仕様書を更新する際にスライドマスターを変更してはいけない、という制約が加わった。

当初、このことは運用上問題なかったが、アプリを拡張するとき、これまでと機能セットの異なる画面の仕様を定義しようとし、これが必要とする画面仕様書の基本的なレイアウトがこれまでの仕様書と異なるために、統合が困難になった。

7.2.2 CI サーバー管理の困難

PowerShell から PowerPoint 等の Microsoft Office アプリを呼び出してファイル操作に使う場合、Office はコンソールを要求する。コンソールが使えないと、PowerShell から Office ファイルを開くことはできても、中身の操作ができない。コンソールが使用可能な状態というのは、すなわちそのアプリのウインドウが画面に表示されているときである。

つまり、Jenkins を用いて自動的にスクリプトを実行する、というサーバーのような使い方をしているのにもかかわらず、画面を表示しておかなければならない。サーバーが再起動したらログインが必要であり、ログインしたらログアウトしてはいけない。

実運用ではノート PC をサーバーにして、自動的に Jenkins ユーザーがログインするように設定したが、PC の蓋を閉じられず、かつ誰でも操作できてしまうので、管理上の大きな制約になった。

8. 総合的な評価

この画面仕様書 CI システムは、運用実績はあるものの、未だ客観的、定量的な評価を行っていない。主観的、および定性的な評価としては、7.1 節で述べた通り、テストの効率化が実現し、開発にも大いにプラスに寄与した。

客観的かつ定量的な評価を行えば、アプリ開発の一環である GUI テストに対しても継続的インテグレーションが役立ち、テストの効率はもちろん、開発全体の効率を上げられるということを示せるはずである。

なお、7.2.1 節で述べた不便さから、アプリの開発が進んだ現在は、このシステムは現在は主力環境としては使用していない。この点を改善し、今後システムを再活用する場合は、アプリ開発の方法

論として昇華させるため、評価面に気を配りたい。

9. 関連するシステム

複数人が同時に画面仕様書を編集できるようなシステムは、オンラインサービスに多い。本システムが採用した PowerPoint には、オンラインで共同編集が可能な Office Web Apps[5] がある。Google Slides[6] は同様のサービスである。より一般的なドローイング向けのサービスには CaCoo[7] があり、さらにワイヤーフレーム作成に特化したサービスには、HotGloo[8]、Pidoco[9] などがある。

仕様書からチェックリストを生成するようなシステムは、本システムのようにスクリプトやプログラムを使えば、どのようなファイル形式に対しても特に問題になることはないはずである。サービスとしてパッケージになっているものについては、現時点では調査不足である。

10. まとめ

画面仕様書を用いて行うテストは簡便であり、工数的なメリットがあるので、さらに精度や効率を上げたい。特に、仕様項目のチェック漏れが発生しやすかったり、画面仕様書を取り回しにくくなったりする弱点がある。これらの弱点に対し、継続的インテグレーションを応用した画面仕様書 CI システムを構築し、テスターが常に最新の画面仕様書とチェックリストを参照できるようにした。

このシステムにより、主観的な評価ではあるが、テストの精度と効率が上がり、ひいては開発全体の効率を上げることができた。今後このシステムを再活用する際は、アプリ開発の方法論とするため、より客観的な評価や、関連するシステムや研究の調査も行っていきたい。

参考 URL

[1] Microsoft PowerPoint
`http://office.microsoft.com/powerpoint`
[2] Microsoft PowerShell
`http://microsoft.com/powershell`
[3] Jenkins CI
`https://jenkins-ci.org/`
[4] Apache Subversion
`https://subversion.apache.org/`
[5] Microsoft Office Web Apps
`http://office.com/webapps`
[6] Google Slides
`http://www.google.com/slides/about/`
[7] Nulab Cacoo
`http://cacoo.com/`
[8] HotGloo
`http://www.hotgloo.com/`
[9] Pidoco
`http://pidoco.com/`