Android application

Testing guideline

第 1.2 版

2013/2/1

Open Embedded Software Foundation

株式会社ソニックス

目次

1. はじめに

ガイドライン策定の背景

Android はオープンソースである事からスマートフォンという分野だけではなく、スマート TV 等のデジタル家電や自動車などに搭載される組込ソフトウェアに至るまで増大の一途をたどっており、その品質が企業の経営のみならず人命や財産に及ぼす影響も大きくなりつつある。その一方でソフトウェア開発の失敗によるデスマーチも話題にこと欠かない。「品質は上流で作り込む」のが品質管理の基本であり、開発における上流工程でのインスペクションやレビューなどの技法や管理方式は従来のソフトウェア開発の物が導入されてきたが、最終的なテストは Android という OS 特有の問題もあり、昨今ではテスト技術に対する反省と関心も高まりつつある。成熟技術ではない Android 開発においてテスト技術に関しては体系的な教育が十分に行われて来たとは言い難く，極めて属人的なものであった事は否定出来ない。

Android アプリケーションの開発は通常のソフトウェア開発と比較すると規模の小さいものが多く、短期間でリリースされる成果物に対してテストの不十分さが指摘されている。
また、開発の下流工程をオフショア企業に委ねることが常態化しつつある現在、テスト工程の品質強化策は重大な意味を持っている。発注者と受注ベンダーという関係のみならず、オフショア企業を含めた開発委託先企業と各国におけるテストの考え方や技術についての誤解を解消し，戦略的テストプロセスやテスト技術の普及・定着を図り、品質を向上することが目的である。本稿では，特に誤解が生じやすく，よって委託先と共通理解を確立しておくべきテストの考え方と技術について述べる。

本ガイドラインの位置付け

Google 社が Android Platform 採用端末に対し実施を義務付けている CTS（Compatibility Test Suite)は、Google Play を含む Market で配布されるアプリケーションが、どの OEM の Android 端末でもスムーズに利用出来る様、主要な公開 API を実行し、チェックを行うテストハーネスである。CTS による Test 群を全て pass する事は互換性を保っているという一つの証になると言われているが、実際にはこの CTS を通っても端末毎の差異が発生する事が起こり得るのが実状である。アプリケーションデベロッパーはこういった端末毎の挙動差異を、開発時に個々に検証をする事が必要になっており、その方針を定めたものは現在、存在していない。

本ガイドラインは Android 向けアプリケーションを開発する際のシステムテストにおいて、どのような観点でテストフェーズを進めるべきかを記したテストガイドラインであり、継続的にアンチパターンを収集・分析し、ガイドラインの改善を行う事でリリース前のアプリケーションの品質の向上、システムテストの精度向上を目的としている。

2.Android Application のテスト項目

テストの目的は，単体テストにて品質が保証されたプログラムが相互に連携して正しく機能するかどうかを検証することである。Android アプリケーションにおいて全てのプログラムが連携して動く事はすなわちリリースされるものと同じ状態である事を示しており、従来のソフトウェア開発における結合テストの位置づけに近いもの、すなわち実機端末で行う事を前提としている。

本章では特に重点的にテストすべき 5 つの項目をテスティングフレームワークとして定義し、そのテスト手法についてどのように行うべきかを記す。

1) ビジネスロジックのテスト

本ガイドラインではビジネスロジックを「各アプリケーションに実装された AndroidOS 及び動作環境ハードウェアに依存しない処理」として定義し、そのテストはアクティビティ（Activity）単位で行う事とする。Android アプリケーションでは UI の表示や各種イベントへの応答等、ユーザとアプリケーション間で行われるやり取りは基本的に全て Activity で管理される事となり、多くのケースでは画面とアクティビティが 1 対 1 に紐づく為、アクティビティ単位でのテストシナリオ（テストケース）を作成する事により、要求仕様として挙げられた全てのビジネスロジックを細分化して網羅する事が出来る為である。

また、システムに依存しないビジネスロジックはテストシナリオの自動化が容易であり、テストツールを用いた自動化により効率化を図る事が可能である。自動化を行うテストスクリプトは Google が提供する公式のテストスクリプトである「NativeDriver」にて作成する事が推奨される。

[本ガイドラインにおけるビジネスロジックテストの基準]
1. テストはアクティビティ単位で実施
2. ビジネスロジックテストはテストスクリプトを用いて自動化する事
3. 自動化は「NativeDriver」を推奨

2) UI のテスト

UI（User Interface）のテストはビジネスロジック同様にアクティビティ単位で行い、テストツールを使った自動化を行い、実行時のテストスクリプトとスクリーンショットをエビデンスとして取得・管理する事を基本とする。テストツールは Nativedriver に対応したもの(※Appendix A.参照)を利用し、テストシナリオ上、自動化出来ない挙動のテストを行う場合には手動でのテスト実施、エビデンスの取得を行う。

[本ガイドラインにおける UI テストの基準]
・画面遷移（フロー）に沿った確認が必要
・エビデンス（スクリーンショット）の取得も必要
・異なる解像度での表示の妥当性

3) 通信に関するテスト

通信を行う機能（処理）を含むアプリケーションの場合、通信関連のテストを行う事は必須である。
但し、どのような通信を行うかは作成するアプリケーションの仕様または使用する回線の種別によって異なる為、本ガイドラインでは最低限行うべき通信関連のテストを下記に記す。

[本ガイドラインにおける通信に関するテストの基準]

１．通信事業者の回線を利用する場合
- **事業者毎の特性を理解し適切なセキュリティ対策が施されていることを確認**
- **通信の特性に対し「受信断」「受信中の帯域変化」「通信途絶」時の各対応を検証する**

２．無線通信回線（WiFi)を利用する場合
- **使用する無線通信の規約に基づく適切なセキュリティ対策が施されていることを確認**
- **この場合特にアプリケーション内で取り扱うデータの保護に留意する**
- **ネットワーク高負荷状態における対応及び未接続、接続断時の各対応を検証**

３．その他通信
- **Bluetooth、NFC 等の規格で通信を行う場合にも取扱うデータの保護に留意する事**

4) デバイス関連のテスト

作成したアプリケーションのテストは基本的にターゲットとしている実端末で行う事が望ましい。テストの方針として、想定した一連の動作を確認し、アプリケーションが動作するハードウェアに依存する処理が実装されている場合はビジネスロジック同様にアクティビティ単位で行う事とする。テストの実行は各アクティビティを１回以上実行するテストシナリオを作成し、自動化する事で開発したアプリケーションを改修する際のデグレーション回避と一連の実行結果をエビデンスとして残し、テストスクリプトを残す事によりコードとテスト両方の継続的な品質の向上が出来るようにする事が最適である。また、テスト実行時に発生するアプリケーションエラーや実機のログ情報も実行時に取れるようにする事も自動化する事のメリットとなる。

[本ガイドラインにおけるデバイス関連テストの基準]
- **アプリケーションの動作確認は実機での動作を前提とする**
- **各アクティビティ（ビジネスロジック）を網羅するテストシナリオを作成する**
- **一連の実行結果をエビデンスとして残し、デグレーションを回避する**

5) 操作／外的要因に対するテスト

Android アプリケーションの実行環境の多くはスマートフォンであることから、電話等の他アプリの割り込み、ホームボタンや電源ボタンの押下、その他ユーザの端末操作に起因する様々な外的要因により、実行中のアプリケーション(Activity)がバックグラウンドへ移動、または破棄されることが頻繁に行われる。そのため、アプリケーションは外部からの処理に影響される事無く正常に動作し、ユーザの操作性や安定性を担保してユーザビリティを下げない事が常に求められる。

ユーザ操作や外的要因に対するテストとしては、Activity がバックグラウンドへ移行する（または破棄される）際に適切な処理が行われているかを確認するために、各画面においてホームボタンや電源ボタンの押下と、その後のアプリケーションへの復帰、端末設定画面からのアプリの強制終了などをテストケースとして行う必要がある。さらに普段のユーザ操作を意識して、端末全体に渡ってアドホックテストを行うことも有効である。

このようなテストケースは、基本的には個々の端末に対し手動での試験を行い網羅していくことが前提となるが、google が提供している「UI Automator」を用いることで、ホームボタン押下からの復帰テスト等を自動化する事もできるため、担保したいユーザビリティによって、テストケースを分類して自動化することで試験を効率的に行うことが可能である。

[本ガイドラインにおける操作／外的要因に対するテストの基準]

・アプリケーションの各画面においてホームボタンや電源ボタン押下、その後のアプリケーション復帰をテストケースとして行う。

・通信など非同期処理を行っている箇所については、データ保持などに留意したテストを実施する。

・普段のユーザ操作に近いアドホックテストを実施する。

3.テスト実施方針

1) テスト計画の策定と実施

Android アプリケーション開発の開発規模は従来のソフトウェア開発と比較すると小規模である場合が多く、また利用するユーザを特定する事が難しい事がある。本項目ではテストを下記３原則によって分類し、各単位におけるテスト計画の策定方法、実施をどのようにするべきかを述べる

[Android アプリケーション開発におけるテスト３原則]

- **TestLevel（単体テスト、結合テスト、受け入れテスト）**
- **TestType（機能性、信頼性、使用性、保守性…）**
- **TestStyle（実機テスト、疑似テスト、自動化テスト）**

定義は各開発担当者により定義されたものである事が望ましい。例えば「TestType」の定義は下記のようなものとする。

（例）TestType の定義
〇機能性（Functionality） 合目的性，正確性，接続性，標準適合性，セキュリティに関するもの
〇効率性（Efficiency） 実行効率性，資源効率性に関するもの
〇信頼性（Reliability） 成熟性，障害許容性，回復性に関するもの
〇使用性（Usability） 理解性，習得性，運用性に関するもの
〇保守性（Maintainability） 解析性，変更性，安定性，試験性関するもの
〇移植性（Portability） 環境適応性，移植作業性，規格準拠性，置換性に関するもの

テスト計画

ソフトウェアの欠陥はそのタイプに応じてそれを検出すべき最適な 3 軸（Level/Type/Style）の組合せが決まるというものである。例えばプログラムの「メモリーリーク」というタイプの欠陥はレベル = 単体テスト，タイプ = 機能性テスト，技法 = 実機または自動化テストで検出するのが最も合理的であり経済的である。このタイプの欠陥が，結合、受け入れテストあるいは稼働後に発症した場合には，再現性の乏しい追及困難な現象となることが多い。テスト計画略立案段階では前述した３原則を元にテスト項目の分類を行いリスクの高い欠陥を特定する事が望ましい。

例)テストシナリオ単位での分類マトリクス

No	Activity名	機能名	テスト内容	level	Type	Style
1	Main	ユーザー覧取得機能	APIを介して全ユーザデータを取得する。	結合	機能性	自動化
2	Main	ユーザー覧表示機能	取得したユーザデータを〇秒以内に表示する。	単体	効率性	自動化
3	Main	ユーザソート機能	XXボタンタップ時、ユーザを〇〇順に並べ替える。	単体	機能性	自動化

テストの実施

テストの実施は事前に策定したテスト計画とテストシナリオケースに基づいて実施する事を前提とする。Android において、アプリケーションが内包する「機能」またはその機能を構成する「処理」を最小単位の単体テスト範囲として捉え、シナリオは単体テストレベルを最小レベルとして作成する。

この最小単位の品質を保証する事が本ガイドラインにおけるテストの目的であり、単体テストは最小単位の（それ以上分解できない）部品に対して実施する事が望ましい。どうしても全てのプログラムの単体テストやコードレビューに投入する工数が捻出できない場合には、入念にテストすべきものと，そうではないものを見極めるための合理的な基準を開発チーム内で設定する。例えば，次のようなプログラムは工数をかけてテストを行う対象となる。

・経験の浅いプログラマが作成したプログラム
・多数のプログラムから引用される使用頻度の高いプログラム
・プログラムステップ数が，ある数を超えるプログラム

2) WEB アプリケーションのテスト

Web ブラウザベースで作成したアプリケーションの場合、Native アプリケーションと比較するとハードウェア性能に依存する問題やバグが出る可能性は少なくなる。但し、現状の Web アプリケーションにおける懸念点の一つであるとされるレスポンス速度は前述した testtype で分類すると「効率性」にあたり、ユーザ入力については「使用性」のテストが必要となる。その為、作成するアプリケーションが WEB アプリケーションである場合にも Native アプリケーション同様のテスト方針の策定並びにテスト自動化を行う事が望ましい。

[Web アプリケーションテストの基準]
・Android Driver を利用したテストの自動化を行う
・テスト 3 原則に応じたテスト方針の策定と実施を行う

3) 多端末テストの実施

Native アプリケーションの場合、フラグメンテーション問題と言われるように端末の特性（画面解像度、バージョン及びカスタマイズ有無）に依存する問題が顕著になることが多い。
そのため、端末特性に依存する様なテストは、対象条件と合致する複数の端末で行う事で端末依存性が問題とならない様品質を保証することが求められる。

しかし、市場に存在するすべての端末を調達し、同じ試験内容を実施することは非常に困難である。
そこで、テストを行う端末を特性毎に分類し、テスト項目の自動化により反復作業を省力化してしまうことが重要となる。ここで、端末依存が顕著になる主要な項目には以下が挙げられる。

[多端末でのテストが必要となる基準]
・AndroidOS バージョン　※対象バージョンを広範囲に指定している場合
・表示画面解像度　※特に Native アプリケーションの場合には解像度の違いによる表示差異の確認が必須
・カメラ等端の内蔵デバイスを利用する場合　※特にアプリケーションが動作保証する範囲の確認
・外部デバイスと連携する場合

アプリケーションの要件から上記を考慮することで、品質を保証するために最低現必要な端末の種類を選定することができる。

また実際の試験実施においては、本紙第２章「Android Application テスト項目」に挙げた観点から、主要端末で手動でのテストを行う項目、自動化を行う項目とを分類することにより、コスト及び検証期間の短縮を図りながら品質の担保が可能となる。

4) テストの自動化と CI

テストは最終的なアプリケーションの品質確認として行うだけでなく、問題を早期に摘出する事で開発の効率性を高める事も意識して実施する事が望ましい。その為には継続的インテグレーション（CI）※を導入し、プロジェクトメンバーがそれぞれ開発した結果を頻繁に結合し、定期的にビルドやテストを行うような環境を作る事が重要である。不具合は早期に摘出し、対応する方が対策費用が抑えられる。本ガイドラインでは前述した通り、Native、Web アプリどちらもテストの自動化を前提としており、実施コストと不具合発生時の対策コストの両方を抑える事で、より、作りこみ段階に充分な工数を割ける方法を推奨する。

※継続的インテグレーション、CI（英: continuous integration）とは、主にプログラマーのアプリケーション作成時の品質改善や納期の短縮のための習慣のことである。エクストリーム・プログラミング (XP) のプラクティスの一つで、狭義にはビルドやテスト、インスペクションなどを継続的に実行していくことを意味する。特に、近年の開発においては、継続的インテグレーションをサポートするソフトウェアを使用することがある。

Appendix A. テストツール

テストの自動実行及び CI においては、以下にあげるツール類を有効活用する事が望ましい。
ガイドラインより各プロジェクトでの開発/テスト計画及び CI の策定を行い、その運用を完遂する事がアプリケーションの安定的な品質確保に繋がる。

[本ガイドラインにおいて推奨するツール]

・Scirocco / Scirocco for Webdriver
http://www.sonix.asia/service/library

・WebDriver
http://seleniumhq.org/projects/webdriver/

・NativeDriver
http://code.google.com/p/nativedriver/

・AndroidDriver
http://code.google.com/p/selenium/wiki/AndroidDriver

・Robotium
http://code.google.com/p/robotium/

・UIAutomator (Provisional)
http://developer.android.com/tools/help/uiautomator/index.html

・SOASTA Test
http://www.soasta.com/testing-solutions/mobile-application-testing/

・Jenkins　（CI）
http://jenkins-ci.org/