OSバージョンアップ　テストカバレージ　ブラックボックステスト

Collaboration Projects

OSバージョンアップ時のアプリケーション試験効率化

移動機開発部　浅野 浩一　増田 真也　小形 充弘　小林 和正

スマートフォン向けOSのバージョンアップに伴うアプリ開発において，OS提供元が公開する技術情報だけでは予測できない不具合を検出するためには試験項目をより多く実施する必要があり，近年のアプリ開発でコスト上昇につながる要因となっている．本研究では，OSバージョンアップ前後の新旧OS間のコードの差分情報を基に，OSバージョンアップで影響のあるアプリコードを試験工程と関連付けたうえで，試験項目ごとに影響のあるアプリコードのテストカバレージ情報を用いてOSコード差分から影響を受ける試験項目を抽出する方式を提案する．

なお，本提案とツールへの実装の取組みでは，株式会社セックをパートナーとし，2015年1月より共同研究を開始している．

1. まえがき

スマートフォン向けOS*1（以下，OS）のバージョンアップに伴うアプリ開発において，OS提供元が公開する技術情報だけでは予測できない不具合に対応するためには試験項目をより多く実施する必要があり，近年のアプリ開発でコスト上昇につながる要因となっている．また，OSバージョンアップの発表から市場へのリリースは短期間になる傾向にあり，バージョンアップ後のOS（以下，新OS）対応アプリのリリースタイミングを新OSのリリースタイミングに近づけることが非常に困難になってきている．そのため，OSバージョンアップへの追従において，OSが提供する新機能やAPI（Application Programming Interface）*2の仕様変更への対応などの，いわゆるバージョンアップ開発を短期間で行うことが重要となり，それは競争力確保のうえで必須である．

バージョンアップ開発におけるコーディング・コンパイル*3といった製造工程では，OS提供元が公開する技術情報資料[1]を基に新機能に対応するためのソースコード（以下，コード）を追加し，既存機能に影響のあるコードの修正などの編集作業を行う．編集作業後はコンパイルなどののち試験工程となるが，その際に，実際にアプリを新OS上で動作させてみると不具合が発生する場合がある．一要因として，技術情報資料には記載されてない動作上の仕様変更が実際には存在していることがある．このため，予期せぬ不具合が発生しているのが実状である．その結果，試験対象範囲を特定できないまま広げていかざるを得ず，ブラックボックステスト*4で行う試験項目数は増加する傾向にあり，近年のアプリ開発期間の長期化とコスト上昇につながっている．

本稿では，ブラックボックステス

*1 **OS**：機器の基本的な管理や制御のための機能や，多くのソフトウェアが共通して利用する基本的な機能などを実装した，システム全体を管理するソフトウェア．

*2 **API**：プログラミングの際に使用できる命令や規約，関数などの集合のことを指す．

*3 **コンパイル**：プログラミング言語で書かれたソースコードを，ヘッダと結合を行ったり文法をチェックしたうえで実行できる形式に変換をすること．

トで行っていた試験項目に対して，試験対象範囲を特定して試験項目を抽出する方式を提案し，試験項目数の削減などその有用性を確認するために試作したAndroid™＊5アプリ向けのシステムの実装方式，実験結果について解説する．

なお，本提案とツールへの実装の取組みでは，株式会社セックをパートナーとし，2015年1月より共同研究を開始している．

2. 提案方式

本研究では，Step 1として，まずアプリ全試験表と開発するアプリケーションのソースコード（以下，アプリコード）を関連付けることでテストカバレージ情報[2]を取得し，Step 2として，バージョンアップ前後のOSコードの新旧バージョン間の差分情報と，開発するアプリコードとを比較することで，OSバージョンアップにより影響があるすべてのアプリコードを抽出する．その後，Step 3として，抽出したアプリコードを試験工程と関連付ける方式を提案する．試験工程はアプリ開発における結合試験・総合試験などに加えて，アプリ開発を発注する側で実施する受入試験なども想定する．本提案方式を用いて，アプリ全試験表からアプリ試験表（抽出版）を作成する手順STEP1～3を**図1**に示しつつ，以下に解説する．

2.1 STEP 1：テストカバレージ情報取得

OSのバージョンアップに先立ち，実施した試験工程のアプリ全試験表（図1①）とアプリコード（図1②）を関連付ける．関連付けの作業は，次の手順で行う．まず，アプリ全試験表の試験項目を実行するためにアプリを走行させる際，アプリコードのどの行を走行させたかを行番号単位で記録する．

ここで記録した内容を試験項目ごとのテストカバレージ情報（図1③）と呼び，**表1**のように表すことができる．表よりソースコードa.java®＊6の20行めを走行させたい場合には，試験項番100を実行すれば良いことが分かる．

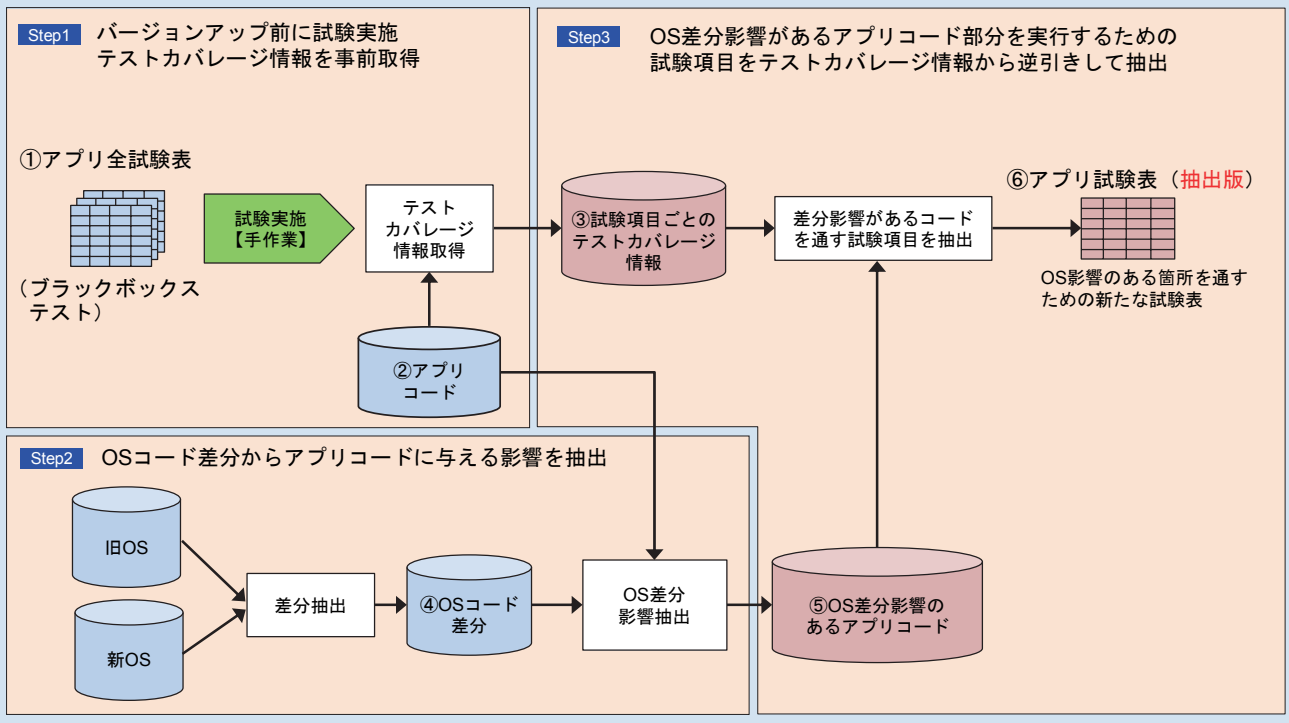


図1 テストカバレージ情報を用いたOS差分影響のある試験項目抽出

＊4 **ブラックボックステスト**：内部の構造を不明としたまま，外部から見た機能を単位として行う評価．結合試験，総合試験，受入試験に用いられることが多い．

＊5 **Android™**：米国Google，Inc.が開発したLinuxベースのオープンプラットフォームで，携帯情報端末を主なターゲットとしている．米国Google，Inc.の商標または登録商標．

＊6 **Java®**：オブジェクト指向のプログラミング言語．Javaにより実装されたアプリケーションは仮想マシン上で実行されるため，異なるプラットフォーム上で動作可能である．OracleとJavaはOracle Corporationおよびその子会社，関連会社の米国およびその他の国における登録商標．文中の社名，商品名などは各社の商標または登録商標である場合がある．

2.2 STEP 2：OS差分影響抽出

OSのコードの新旧バージョン間を差分抽出すると**表2**のように表すことができる．これをOSコード差分（図1④）と呼ぶ．例えばOSコードのDEF.javaではバージョンアップ前後でAPIの内部処理が変更されており，アプリで呼び出した際に動作に差分がある可能性があることが分かる．OSコード差分とアプリコードを突き合わせることでOS差分のアプリへの影響を抽出する．アプリコードを**表3**とした場合，OSコード差分により影響のあるアプリコード（図1⑤）を**表4**のように表すことができる．ここからアプリコードa.javaの20行めは，OSバージョンアップの影響を受けることが分かる．

2.3 STEP 3：試験項目抽出

STEP 1で記録した試験項目ごとのテストカバレージ情報（図1③）に対してSTEP 2で抽出したOS差分影響のあるアプリコード（図1⑤）を突き合わせて，アプリ試験表（抽出版）（図1⑥）を抽出する．表1と表4から，**表5**が求められる．アプリ全試験表にある試験項目のうち，試験項番100を実行すればOSバージョンアップの影響があるアプリコードを走行する試験として十分であり，試験項番200は今回のOSバージョンアップでは実行不要であることが分かる．

3. 実装方式

3.1 テストカバレージ情報の取得環境と取得情報の整形方法

⑴取得環境

テストカバレージ情報は，AndroidStudio*7 [3]に組み込まれているJaCoCo（Java Code Coverage Library）*8 [4]を使用して取得する．AndroidStudioでは，InstrumentationTest*9 [5]による試験実行時にその取得が可能となっているが，試験実行中に操作用PCとAndroid端末をADB（Android Debug Bridge）*10 [6]で接続し続ける必要があり，ローバッテリ状態を維持する必要がある試験項目の実行時に給電されてしま

表1　試験項目ごとのテストカバレージ情報（例）

試験項番	ソースコード	実行済行番号（テストカバレージ情報）
100	a.java	20
200	b.java	30

表2　OSコード差分（例）

ファイル名	変更前（Ver 5.0）	変更後（Ver 6.0）	変更差分
ABC.java	int ABC(a, b, c){	int ABC(a, b, c, d){	パラメータが増加
DEF.java	g = defexec();	g = def2exec();	内部処理が変更

表3　アプリコード（例）

ファイル名	行番号	ソースコード内の命令文
a.java	10	int r = 0;
	20	ret = DEF();
b.java	150	log(ABC(a, b, c));

表4　OS差分影響のあるアプリコード（例）

ファイル名	行番号	種　別	影響内容
a.java	20	警　告	呼出し関数DEFの内部処理が変更された
b.java	150	致命的	API ABCのパラメータの数が増加された

＊7 **AndroidStudio**：Androidアプリケーションの統合開発ツール．
＊8 **JaCoCo**：Javaソースコードのテストカバレージを取得するためのライブラリ．
＊9 **InstrumentationTest**：Androidアプリで自動試験を行うための仕組み．

うなど，一部の試験項目実行の妨げとなっていた．そこで，InstrumentationTestに任意のタイミングでテストカバレージ情報のリセット*11，ダンプ*12が行える機能を追加し，試験実行時のADBでの接続を回避できるようにした．**図2**に，構築した環境と実際の手順を示す．試験実施者は，試験手順実施前にリセットを行い，JaCoCoが収集したテストカバレージ情報を削除する．試験手順を実施中はJaCoCoがテストカバレージ情報をメモリ内に収集し，その後収集したテストカバレージ情報をストレージにダンプする．

表5　アプリ試験表（抽出版）（例）

試験項番
100

⑵取得情報の整形方法

テストカバレージ情報は，ある試験項目を実行した際に実行したアプリコード一覧として取得される．実行した全試験項目分のテストカバレージ情報を，アプリコード行番号をキーにマージし，試験実行時に通過するアプリコード一覧（**表6**）を得る．

3.2　OSのコード差分の抽出手順とOSコード差分から影響を受けるアプリコードの抽出手順

OSのコード差分を，バージョンアップ前後のOSソースコードから，**図3**に示すフローに従って，差分のあるクラス*13，メソッド*14の影響を受けるクラス・メソッドの一覧として抽出する．

OSのコード差分（クラス，メソッド）を使用している部分をアプリコード内から抽出し，OSのコード差分から影響を受けるアプリコード一覧（**表7**）を得る．

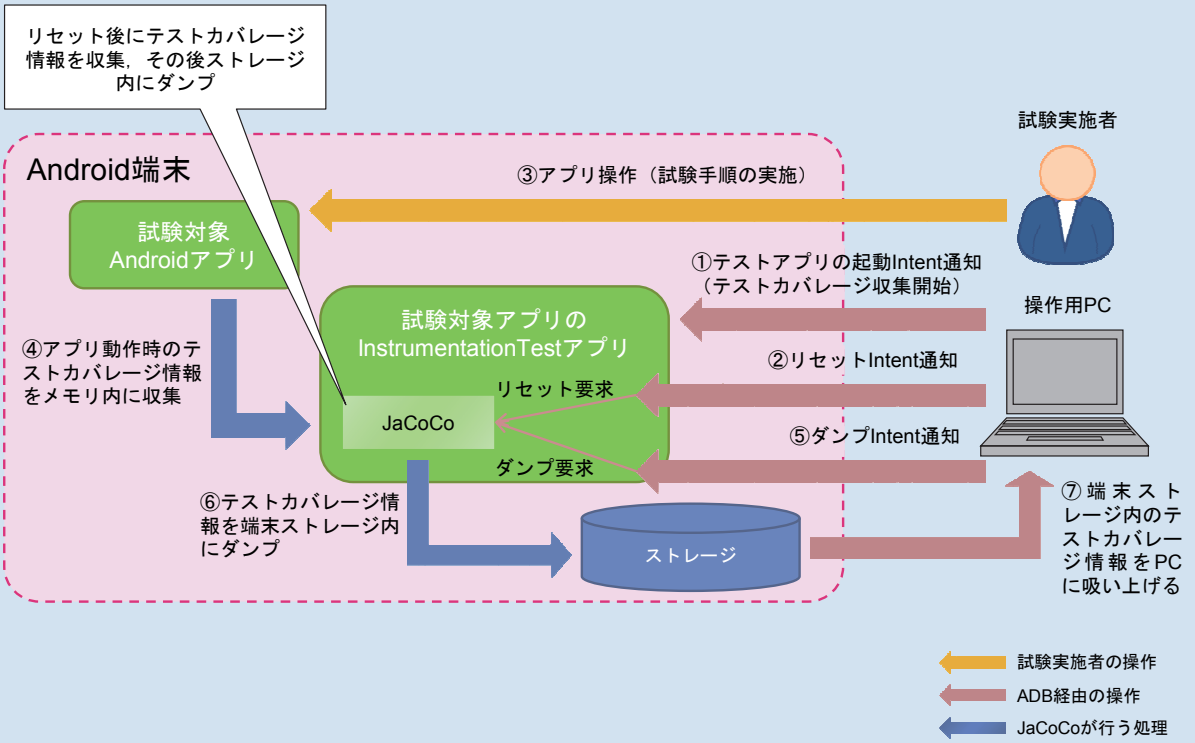


図2　試験実施時のテストカバレージ情報取得環境および取得手順概略

＊10 ADB：Android SDKに含まれるツール．シェルコマンドの発行やファイル転送などが行える．

＊11 リセット：ストレージ内に溜めていたテストカバレージ情報を削除する操作．他の試験項目で記録するテストカバレージ情報と混同しないために行う．

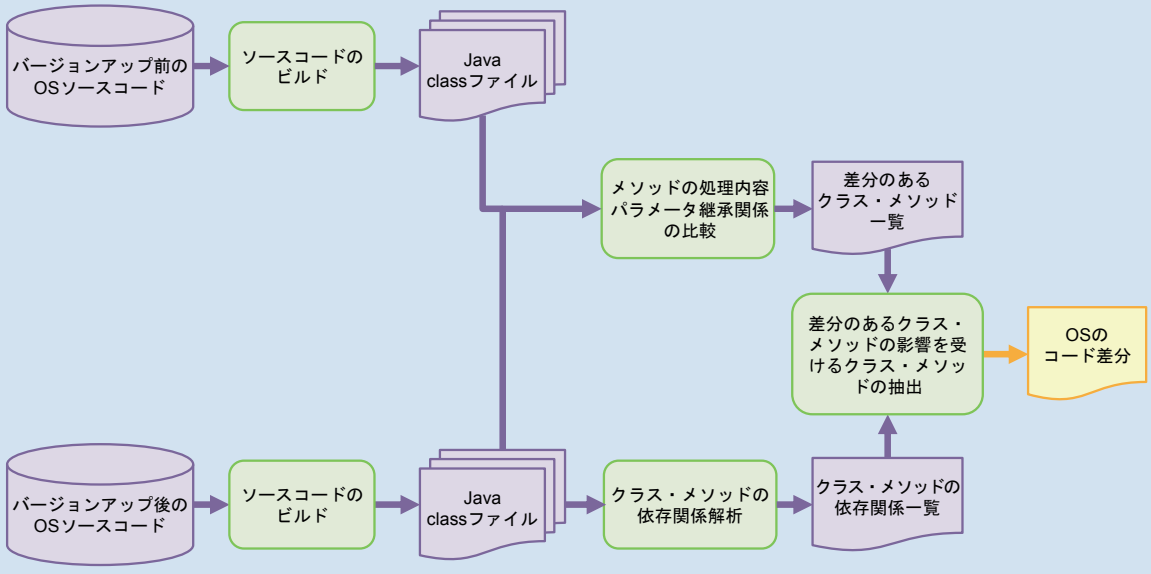
＊12 ダンプ：テストカバレージ情報をストレージに溜める操作．試験項目ごとに，ダンプしたテストカバレージ情報を記録して利用する．

＊13 クラス：オブジェクト指向プログラミングにおける，同様な状態や振舞いを持つオブジェクトをまとめた1つの型のこと．

＊14 メソッド：オブジェクト指向プログラミングにおける，オブジェクトがもっている振舞いのこと．

表6　試験項目実行時に通過するアプリのコード一覧（例）

アプリコード		試験項番1	試験項番2	…	試験項番X
ソースファイル名	行番号				
AAAAAA.java	10	○			○
	20	○	○		
	100		○		
BBBBBB.java	5		○		○
	15	○			○
	25	○			
⋮					



ビルド：ソースコードを実行可能なファイルに変換する作業．Javaコンパイラにより行われる．

図3　OSのコード差分抽出フロー

表7　OSのコード差分から影響を受けるアプリのコード一覧（例）

アプリコード		OS差分影響Lv	影響内容
ソースファイル名	行番号		
AAAAAA.java	10	3	API返却値のクラス実装変更
	100	5	APIパラメータの変更
BBBBBB.java	15	4	API内部ロジックの変更
	50	3	コンストラクタのロジック変更
	100	4	APIのthrows指定追加
⋮			

3.3 OSのコード差分から影響を受ける試験項目の抽出手順

表6，および表7で得られた一覧を，アプリコード行番号をキーにマージし，OSのコード差分から影響を受ける試験項目の一覧（**表8**）として抽出する．

4. 実　験

本研究の手順を，5.1.1から6.0.0へバージョンアップした際のAndroidアプリと，そのアプリの受入試験項目に適用し，試験範囲が特定できているか，またそれにより実行する試験項目がどれだけ削減できたかを計測した．

4.1 実験結果

本研究の手順を適用してOSのコード差分に影響を受けるアプリコード，試験項目を抽出した結果を**表9**に示す．

実験により抽出した試験項目は既存方式と同等の精度で抽出することができ，提案方式により不具合検出に繋がる項目を漏らすことなく試験範囲が特定できていることが分かった．一方で，4アプリ中2つのアプリでは削減項目0件，残りの2アプリでも削減項目数は2～4件（削減効果は2～4％）と，当初期待していた試験範囲の削減効果を得ることができなかった．

これは，初期化処理や画面のベースクラスの処理など，どの試験項目を実行しても通過するソースコード行が存在しており，この部分にOS影響があったため，ほぼすべての試験項目が“影響あり”と判断されてしまったことが原因である．

4.2 課題への対応

この問題点を回避するため，アプリコードと試験項目の間に次のような関係があると仮定した．

- 仮定1：どのような試験項目を実行しても通過するアプリコード行については，どれか1項目の試験を実行して当該行の動作が確認できればよい．
- 仮定2：特定の試験項目を実行した場合のみ通過するアプリコード行については，その試験

表8　OSのコード差分から影響を受ける試験項目の一覧（例）

アプリコード		OS差分影響有無	試験項番1	試験項番2	…	試験項番X
ソースファイル名	行番号					
AAAAAA.java	10	有	○			○
	20		○	○		
	100	有		○		
BBBBBB.java	5			○		○
	15	有	○			○
	25		○			
⋮						

表9　実験結果

アプリ名	コード行数（Line）		試験項目数	
	OS差分影響あり	全　体	OS差分影響あり	全　体
災害用キット	1,676	7,964	28	28
スケジュール＆メモ	11,208	75,125	19	19
はなして翻訳	8,548	41,844	165	169
音声UI	1,025	4,323	47	49

項目を実行して当該行の動作を確認する必要がある．

上記の仮定に基づき，実行すべき試験項目を削減する手順の概略例を**表10**に示す．1つの試験項目でしか実行されていないアプリコード行を赤枠で囲った．

また，この手順を実験結果に適用し，実施必須の試験項目数のみに削減した結果を**表11**に示す．それぞれのアプリの削減後の試験項目数を赤枠で囲った．

全体の試験項目数が少ない「スケジュールメモ」は削減率が低いものの，その他の3アプリについては7割以上の削減効果が出ており，OSバージョンアップ時の影響を確認する試験の効率化（＝実行する試験項目の削減）に十分な効果があった．また，手動で試験項目を実行した結果と比較したところ，削減した試験項目の中に，手動で実行した際に不具合を発見した試験項目は含まれていなかった．仮定1，2を基に削減された項目の品質への影響の妥当性は未検討であるが，今回の4つのアプリへの適用では，試験品質を維持したまま，実行する試験項目の削減できた結果となった．

4.3　今後の展望

OSのコード差分から影響を受ける試験項目の一覧（表8）を抽出した際，OSバージョンアップ時の影響があるにも関わらず，既存方式において試験が実施されていないアプリコードも抽出可能であることが判明した．実際に，今回の実験にて得た結果から試験未実施となっているアプリコードを抽出した結果を**表12**に示す．それぞれのアプリでOS差分の影響があるものの試験未実施であるアプリコード行数を赤枠で囲った．この方法を応用すれば，他のアプリにおいても試験項目の不足を検

表10　試験項目削減方法概略（例）

アプリコード		試験項番1	試験項番2	試験項番X
ソースファイル名	行番号			
AAAAAA.java	11	○		○
	12	○	○	
	13		○	
BBBBBB.java	21		○	○
	22	○		○
	23	○		
⋮				
項目削減可否		不可 仮定2に基づき実行必須	不可 仮定2に基づき実行必須	可 仮定1に基づき，他の試験項目実行で代替できる

1つの試験項目でしか実行されていないアプリコード行

複数の試験項目で実行されているアプリコード行

表11　項目削減後の実験結果

アプリ名	試験項目数（項目）			削減率
	実施必須 項目数	OS差分 影響あり	全　体	
災害用キット	18	126	126	85.7%
スケジュール＆メモ	16	19	19	15.8%
はなして翻訳	45	165	169	73.4%
音声UI	6	47	49	87.8%

表12　OSバージョンアップの影響を受ける試験未実施行数

アプリ名	コード行数（Line）		
	OS差分影響あり		全　体
	試験 未実施	試験 実施済み	
災害用キット	790	886	7,964
スケジュール＆メモ	8,075	3,133	75,125
はなして翻訳	5,503	3,045	41,844
音声UI	0	1,025	4,323

出することが可能である．

なお，今回実験の対象としたOSバージョンアップでは，提案方式による試験項目削減を行ったとしても市場に流出した不具合は0件であったが，今後のOSバージョンアップに向けて，試験項目削減とは逆の方向性ではあるが，ソフトウェア品質の向上の余地があるかについても検証を行いたい．

5.　あとがき

本稿では，OSバージョンアップの影響を受けるアプリの試験項目を抽出する方式を提案し，試作システムにより動作検証を行うことでアプリ試験の効率化を確認した．

今後は，新OSがリリースされた際に本提案方式を活用し，適用するアプリの数を増やしていくことで試験項目の削減と新規項目の作成漏れの防止につなげていく．また，自動化試験においてもより多くのケースにおいて正しく動作する実装方式を検討し，本提案手法の活用範囲を広げていく予定である．

文　献

[1] Android Developersホームページ．https://developer.android.com/index.html

[2] 保田 勝道：“ソフトウェア品質保証の考え方と実際―オープン化時代に向けての体系的アプローチ―，”日科技連出版社，1995.

[3] AndroidStudio：“Android Studioの概要．”https://developer.android.com/studio/intro/index.html

[4] EclEmma：“JaCoCo Java Code Coverage Library.”http://www.eclemma.org/jacoco/

[5] Android Studio：“Test Your App.”https://developer.android.com/studio/test/index.html

[6] Android Studio：“Android Debug Bridge.”https://developer.android.com/studio/command-line/adb.html