ミューテーションテストを利用した遺伝的アルゴリズムによるAndroidアプリケーション用テストスイート生成

嶌津 達也[1]　高田 眞吾[1]　倉林 利行[2]　丹野 治門[2]

概要：
現在 Android アプリケーションの数は約 220 万と非常に膨大な一方で，質の低いアプリも多いという現状がある．そのため，Android アプリに対する自動テストは現在盛んに研究されているが，実際には開発者の多くが手動でしかテストをしていない．既存の Android アプリ向け GUI テストツールの中では Monkey，Dynodroid，EvoDroid が高いカバレッジを達成している．本稿では EvoDroid が適応度関数にカバレッジを用いている点に着目し，カバレッジよりも実際のバグと相関が高いと言われているミューテーションテストを利用することを提案する．

キーワード：Android，遺伝的アルゴリズム，ミューテーションテスト

1. 序論

現在スマートフォン市場は非常に大きい市場となっている．その中でも Android OS は世界シェアが 1 位であり，それに伴い Android アプリケーション（以下，Android アプリと呼ぶ）の数も 2016 年 6 月時点で 220 万本と増え続けている [1]．しかし，その一方で質の低いアプリも多く存在するという現状がある．アプリの開発者の多くはテストを手動で行っており，自動テストを用いていたとしてもランダムテストが主流となっている．そのため近年では Android アプリに対する GUI テストの自動化に関する研究が盛んに行われている．

Android アプリに対する GUI テストは近年様々な研究がされている．Google 社は GUI のランダムテストを自動実行する Monkey [2] を提供している．Machiry ら [3] は Android アプリの動作を監視して，動的に GUI テストを実行する Dynodroid を提案した．Choudhary ら [4] は Monkey や Dynodroid，その他 5 つの Android 用テストツールの性能比較を行っており，総合的に Monkey と Dynodroid が最も良い結果を示した．Mahmood ら [5] は遺伝的アルゴリズムを用いて Android アプリに対するテストシナリオ（一連のテストシーケンス）を生成する EvoDroid を提案した．EvoDroid では適応度関数にカバレッジを用いてテストシナリオを生成しており，Monkey や Dynodroid と比べて高いカバレッジを達成している。しかし，Just らの研究 [6] ではカバレッジよりもミュータントの方が実際のバグと相関が高いという結果が示されている．

そこで本研究では EvoDroid が遺伝的アルゴリズムを用いて高いカバレッジを達成したことに着目し，遺伝的アルゴリズムの適応度関数にミューテーションテストを利用することを提案する．カバレッジと比べて実際のバグと相関が高いミュータントを基準として取り入れることで，バグ検出能力の高いテストスイートの生成ができると考えられる．

以降，2 章では Android アプリの概要と本研究の要素技術となる遺伝的アルゴリズムとミューテーションテストについて述べる．3 章では Android アプリを対象とした GUI テスト，及びミューテーションテストに関する研究について述べる．4 章では本研究で提案するツールとそれを構成する各モジュールの詳細について述べる．そして，5 章で結論と今後の課題を述べる．

2. 背景

本章では Android アプリの概要，及びテストシナリオの生成に用いる 2 つの技術，遺伝的アルゴリズムとミュー

[1] 慶應義塾大学
Keio University
[2] NTT ソフトウェアイノベーションセンタ
NTT Software Innovation Center

テーションテストについて述べる．

2.1 Android アプリ

Android アプリは Java 言語で記述されており，主に Android Manifest ファイルと以下の 4 種類のコンポーネントで構成される．

- Activity
  XML で書かれたレイアウトデザインを基に画面を描画し，GUI の機能を制御する．レイアウトは View という Widget を定義し，GUI のデザインを制御する．
- Service
  音楽の再生など UI の処理を伴わないバックグラウンドのタスクの処理を行う．
- Broadcast Receiver
  電池残量の通知などシステムレベルの情報を受け取り，処理を行う．
- Content Provider
  ファイルシステムのデータベースからのデータ取得，データ保存に用いられる．

また，UI のレイアウトは XML で記述され，res フォルダに置かれる．

2.2 遺伝的アルゴリズム

遺伝的アルゴリズムとは自然界のシステムの適応過程を模倣し，生物の進化のメカニズムを模したアルゴリズムである．遺伝的アルゴリズムの一般的な手順は以下の通りである．

(1) ランダムな個体を N 個生成し，初期個体群として設定する．
(2) 各個体の適応度を計算する．
(3) 設定された選択方法によって個体の選択を行う．適応度が低い個体ほど淘汰されやすい．
(4) 設定された交叉率，交叉方法により交叉を行い，新しい個体を生成する．
(5) 設定された突然変異率，突然変異方法により突然変異を行い，新しい個体を生成する．この結果，新しい世代の個体群が生成される．
(6) 終了条件を満たしていれば，その時点で得られている最良の個体を準最適解とし，終了する．終了条件を満たしていなければ，2 に戻る．

本研究では遺伝的アルゴリズムを利用してメタヒューリスティックにテストシナリオの生成を行う．

2.3 ミューテーションテスト

ミューテーションテストはテスト対象のプログラムに対して人工的にバグを埋め込むことで，テストスイートの質を評価する手法である．ミューテーションテストの一般的な手順は以下の通りである．

(1) テスト対象プログラムとテストスイートを用意する．
(2) テスト対象プログラムにバグを埋め込み，ミュータント（バグ入りプログラム）を生成する．
(3) 元のプログラムとミュータントに対してテストスイートを実行する．
(4) 実行結果を比較する．

元のプログラムとミュータントの実行結果を比較した際，結果が異なっていればテストが埋め込まれたバグを検出できたとみなす．これを “テストがミュータントを kill する” と呼ぶ．もし同じ結果であればテストはミュータントのバグを検出できていないことになるので，テスト改善の余地があると考えられる．

2.3.1 ミューテーションオペレータ

プログラムに人工的なバグを埋め込むことをミューテーションといい，バグを埋め込む際の規則のことをミューテーションオペレータという．ミューテーションオペレータは例えば＋をーに変更するといった変更をソースコードに 1 箇所だけ埋め込む．ミュータントの生成ではミューテーションオペレータが適用可能な箇所それぞれにミューテーションを行い，その数だけミュータントが生成される．

2.3.2 ミューテーションスコア

ミューテーションテストにおける評価指標として一般的にミューテーションスコアが用いられている．ミューテーションスコアは以下の式で表される．最大値は 1 であり，そのときテストは生成したミュータントをすべて kill できる．

$$mutation\ score = \frac{|killed\ mutants|}{|all\ mutants|}$$

ミューテーションテストはテストがミュータントを kill できるならば，そのテストは現実の同様のバグも検出できるであろうという考え方に基づいている．

本研究では遺伝的アルゴリズムの適応度関数においてミューテーションテストの考え方を用いる．各テストシナリオのミューテーションスコアを計算して適応度として用いることで，バグ検出能力の高いテストシナリオの遺伝子を次世代へ引き継ぐことができる．

3. 関連研究

本章では Android アプリを対象とした GUI テストに関する研究，及び Android アプリを対象としたミューテーションテストに関する研究について述べる．

3.1 Android アプリに対する GUI テスト

Android アプリに対する GUI テストは近年様々な研究がされている．Aravind Machiry らは動的にテストイベントを生成する Dynodroid というツールを提案した [3]．Dynodroid はイベントを実行するごとにテスト対象アプリ

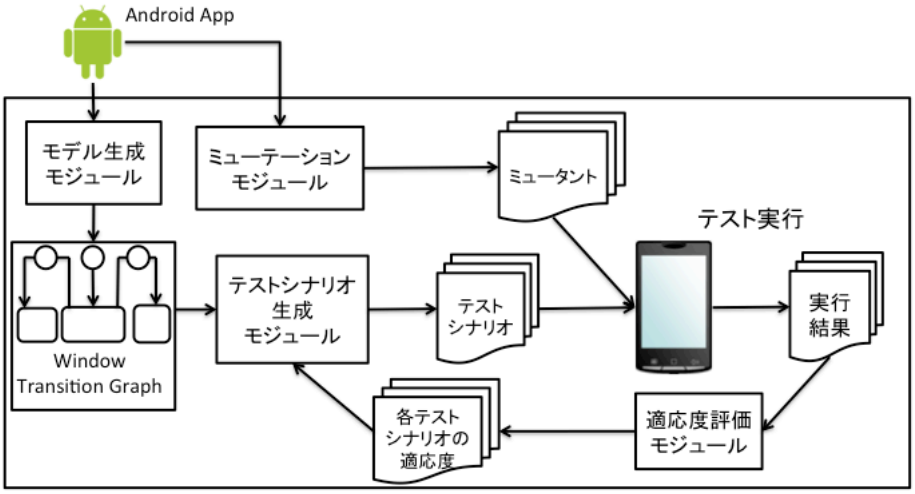

図 1　ツールアーキテクチャ

の GUI Hierarchy から次のテストイベントを生成する．

Riyadh Mahmood らは遺伝的アルゴリズムを利用してテストを生成する EvoDroid というツールを提案した [5]．EvoDroid では適応度関数にカバレッジを用いることで Monkey や Dynodroid よりも高いカバレッジを達成している．本研究では適応度関数にミューテーションテストを用いることで実際のバグ検出能力に焦点を当ててテスト生成を行う．

Shauvik Roy Choudhary らは Android アプリの GUI テストツールである Monkey[2]，ACTEve[7]，GUIRipper[8]，Dynodroid[3]，A^{3}E[9]，SwiftHand[10]，PUMA[11] の 7 種類について比較実験を行った [4]．ツールの使いやすさ，Android OS の複数のバージョンへの適用，コードカバレッジ，fault 検出能力の 4 つの観点から評価を行い，総合的に Monkey と Dynodroid が良い結果を示した．

3.2　Android アプリにおけるミューテーションテスト

Deng Lin ら [12] は Android アプリ特有のミューテーションオペレータを提案した．Deng らは Android アプリを構成する Java ファイルと XML ファイルに対する計 8 種類のミューテーションオペレータをテスト評価能力の観点から既存のミューテーションオペレータと比較を行った．

現在，Android アプリに対するミューテーションテストに関する研究はあまり行われていない．この理由としてミューテーションテストが基本的に単体テストレベルにおける手法であることが挙げられる．Android アプリは Java と XML ファイルで構成されているので，単体テストにおいては既存のミューテーションテスト技術で事足りるからであると考えられる．

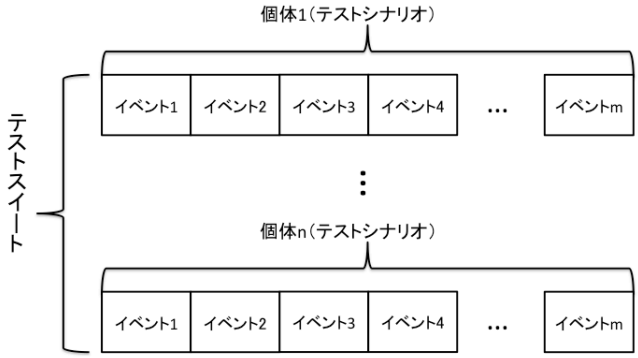

図 2　テストスイート，テストシナリオ，イベントの関係

4.　提案ツール

本章では遺伝的アルゴリズムとミューテーションテストを利用して Android アプリケーション用のテストスイートを生成するツールについて述べる．本提案ツールのアーキテクチャを図 1 に示す．

遺伝的アルゴリズムにおける各用語と本研究における用語の対応は以下の通りである．

- 集団（個体群）：テストスイート
- 個体：テストシナリオ
- 遺伝子：イベント

テストスイートは複数のテストシナリオから成り，各テストシナリオは複数のイベントから成る（図 2）．

本提案ツールのテストスイート生成の流れは以下の通りである．

(1) テスト対象の Android アプリのプロジェクト（ソースコード）を用意する．

(2) モデル生成モジュールにより，GUI の情報や Activity の遷移情報を含んだ Window Transition Graph（WTG）

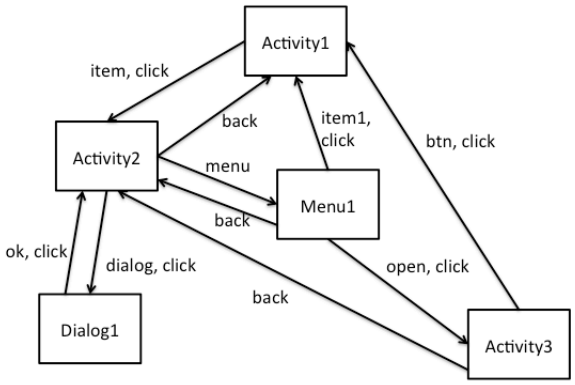

図 3 Window Transition Graph

を生成する．

(3) ミューテーションモジュールにより，ミュータントアプリを生成する．

(4) テストシナリオ生成モジュールにより，WTG の情報を基に初期世代となるテストシナリオを生成する．

(5) 元のアプリ，ミュータントアプリ，テストシナリオをエミュレータにインストールし実行する．

(6) 得られた結果から適応度評価モジュールによって各テストシナリオの適応度を計算する．

(7) 各テストシナリオの適応度を基にテストシナリオ生成モジュールで次世代のテストシナリオを生成する．

(8) 設定された終了条件を満たすまで 5〜7 を繰り返す．

4.1 モデル生成モジュール

モデル生成モジュールには GATOR[13] を用いる．GATOR で生成される WTG は以下の情報を持つ．

- Window（Activity，Menu，Dialog）
- 遷移を引き起こすイベントとそのイベントハンドラを持つ View
- Activity の遷移関係

WTG の情報を基に初期世代のテストシナリオの生成，及び遺伝的アルゴリズムにおける交叉等各種の操作を行う．図 3 に WTG の例を示す．例えば，Activity1 において item という View をクリックすると Activity2 に遷移する．

4.2 ミューテーションモジュール

ミューテーションモジュールには muJava [14] を拡張したものを用いる．muJava はミューテーションを行った Java ファイル（ミュータント）を生成する．本研究では拡張により，その Java ファイル（ミュータント）を含んだプロジェクトをビルドし，Android アプリのソフトウェアパッケージである apk ファイル（ミュータントアプリ）を生成する．

ミューテーションモジュールでは以下の流れでミュータントを生成する．

(1) Android プロジェクトからミューテーションの対象となる Java ファイルを選択する．

(2) muJava の GUI から適用するミューテーションオペレータをテスタが選択する．

(3) 選択されたミューテーションオペレータに従って Java ファイルにバグを埋め込み，バグ入りの Java ファイルを生成する．

(4) バグ入りの Java ファイルとその他のソースコードを一緒にビルドし，ミュータントアプリ（apk ファイル）を生成する．

なお，バグの埋め込み規則であるミューテーションオペレータは muJava（Version 4）にデフォルトで実装されているものを用いる．

4.3 テストシナリオ生成モジュール

テストシナリオ生成モジュールでは遺伝的アルゴリズムを利用してテストシナリオの生成，実行を行う．テストシナリオ生成モジュールの実行の流れは以下の通りである．

(1) 生成した WTG を基に，テストシナリオを N 個体ランダム生成する．生成では WTG から各 Activity におけるイベントと View の情報を抽出し，その中からランダムにイベントと対象となる View を選択する．イベントを行った際 Activity が遷移する場合，次のイベント選択時にはその遷移先の Activity に対するイベントの中から選択する．これを任意の回数 M 回繰り返すことで，長さ M のテストシナリオを生成する．

(2) 各テストシナリオと元のプログラム，ミュータントアプリをエミュレータにインストールし実行する．

(3) テストスイート（テストシナリオ群）のミューテーションスコアを基に終了条件を満たすか確認する．

(4) 各テストシナリオによるミュータントアプリの kill 情報を適応度評価モジュールに渡す．

(5) 適応度評価モジュールから各テストシナリオの適応度の値を受け取る．

(6) 遺伝的アルゴリズムに従って以下の操作を行い，次世代のテストスイートを生成する．

- 各テストシナリオの適応度を基に選択を行う．
- 設定された交叉率で交叉を行う．
- 設定された突然変異率で突然変異を行う．

(7) ステップ（2）に戻る．

次に，終了条件，選択，交叉，突然変異について詳しく述べる．

4.3.1 終了条件

テストシナリオ生成モジュールにおける遺伝的アルゴリズムの終了条件は以下の通りである．

現世代のミューテーションスコアと前世代のミューテーションスコアを比較し，次のようにする．

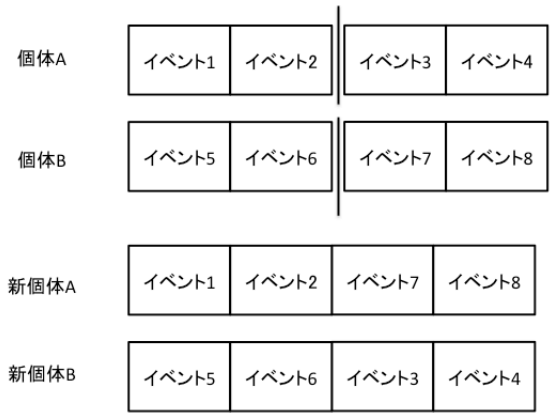

図 4　一点交叉

- 上回った場合：実行を続ける．
- 同じか下回った場合：テスタが終了するか判断する．

現世代のスコアが前世代のスコア以下の場合，それ以上実行してもスコアは改善しない可能性がある．そのためテスタが現世代のスコアを確認し，すでにスコアが十分高いと判断した場合は実行を打ち切り，スコアが十分でないと判断した場合は実行を続ける．また，スコアが改善しない状態が何世代も続き，テスタがこれ以上実行してもスコアは改善しそうにないと判断した場合は実行を打ち切る．

また，ミューテーションスコアが 1 に到達，つまり生成されたミュータントをテストスイートが全て kill できた場合は実行を終了する．

4.3.2　選択

選択では適応度を基にどの個体を次世代へ残すかを決定する．本研究ではエリート選択とトーナメント選択を用いて，選択を行っている．この選択では初期世代の個体数 N に到達するまで選択を行う．

エリート選択とは現世代で最も適応度の高い個体を必ず残す選択手法である．エリート選択で選択された個体は 4.3.3，4.3.4 小節で行う操作の対象として選ばれない．つまり，その時点での最良の個体は必ず次の世代に残す．

トーナメント選択とは現世代から任意の数だけ個体をランダム選択し，その中で最も適応度の高い個体を選択する手法である．トーナメント選択を行うことによって著しく適応度の高い個体が現れても局所最適解に陥りにくくなる．

4.3.3　交叉

交叉では 2 個体の遺伝子を入れ替えて新たな個体を生成する．本研究では一点交叉を用いて交叉を行う．一点交叉とは図 4 のように交叉点をランダムに 1 箇所選び，その交叉点よりも後ろの遺伝子を入れ替える方法である．この例は交叉点を 2 とした場合である．

本研究では交叉を行う際，モデル生成モジュールで生成した WTG の情報を利用し Window を考慮する．これは交叉を行った結果実行不可能なテストシナリオの生成を防ぐためである．具体的な交叉手順としては，まず 1 つめの個体において交叉点をランダムに選択する．その交叉点に対応する Window を WTG から抽出する．次に，2 つめの個体においてその Window と対応する交叉点を抽出し，その中からランダムに 1 つ選択する．そして，それぞれの個体で選ばれた交叉点より後ろの遺伝子（イベント）を入れ替える．

4.3.4　突然変異

突然変異では個体の遺伝子の一つを別の遺伝子に書き換える．本研究ではテストシナリオを構成するイベントに対して以下の 4 つのうちのどれかの操作を行う．

- 追加
  追加操作はランダムな位置に新たなイベントを挿入する．選択された位置に対応する Activity を WTG から抽出し，その Activity に対するイベントをランダムに選択する．
- 変更
  変更操作はランダムにイベントを一つ選択し，別のイベントで置き換える．変更操作においても新たなイベントは WTG の情報を基に生成する．
- 削除
  削除操作はランダムにイベントを一つ選択し，そのイベントを削除する．ただし，実行不可能なテストシナリオの生成を防ぐため，Window の遷移を伴うイベントは選択対象から除く．
- スワップ
  スワップではランダムに 2 つのイベントを選択し，位置を入れ替える．選択するイベントは同じ Window に対するイベントのみを選択する．

4.4　適応度評価モジュール

適応度評価モジュールでは各テストシナリオの適応度を計算する．適応度が高いほど選択操作において選択される確率が上がり，適応度が低いほど選択される確率が低くなる．つまり，適応度が高い個体ほど次世代に残りやすくなり，適応度が低い個体ほど淘汰されやすくなる．

遺伝的アルゴリズムを用いたテスト生成ツールである EvoDroid[5] では適応度関数にパスカバレッジを用いている．しかし，Just らの研究 [6] ではカバレッジよりもミュータントの方が実際のバグと相関が高いという結果が示されている．そこで，本研究では適応度関数にミューテーションテストの考え方を用いることで，バグ検出能力の高い個体を残す．これにより，既存ツールよりもバグ検出能力が高いテストの生成ができると考えられる．

適応度は以下の式で計算される．

$$f(t_i) = \frac{1}{n}\sum_{j=1}^{n} x_{ij} \qquad x_{ij} = \begin{cases} 1 & (t_i\ kills\ m_j) \\ 0 & (t_i\ can't\ kill\ m_j) \end{cases}$$

この適応度関数ではより多くのミュータントを kill 可能

なテストシナリオほど高い適応度となる．

5. 結論

本論文では遺伝的アルゴリズムとミューテーションテストを用いた Android アプリ用テストスイート生成手法を提案した．テスト生成にミューテーションテストを利用することにより，他のテストツールと比べて高いバグ検出能力を持つテストの生成ができると考えられる．

今後の課題は Android 特有のミューテーションオペレータの実装やケーススタディによるツールの評価，設定変数である交叉率，突然変異率，集団サイズを決定するための予備実験を行うことが挙げられる．

参考文献

[1] AppBrain: Google Play stats, `http://www.appbrain.com/stats`.

[2] Google: Monkey, `https://developer.android.com/studio/test/monkey.html`.

[3] Machiry, A., Tahiliani, R. and Naik, M.: Dynodroid: an input generation system for Android apps, *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, pp. 224–234 (2013).

[4] Choudhary, S. R., Gorla, A. and Orso, A.: Automated test input generation for Android: Are we there yet?, *Proceedings of 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pp. 429–440 (2016).

[5] Mahmood, R., Mirzaei, N. and Malek, S.: EvoDroid: segmented evolutionary testing of Android apps, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, ACM, pp. 599–609 (2014).

[6] Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R. and Fraser, G.: Are mutants a valid substitute for real faults in software testing?, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, pp. 654–665 (2014).

[7] Anand, S., Naik, M., Harrold, M. J. and Yang, H.: Automated Concolic Testing of Smartphone Apps, *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, ACM, pp. 59:1–59:11 (2012).

[8] Amalfitano, D., Fasolino, A. R., Tramontana, P., Carmine, S. D. and Memon, A. M.: Using GUI Ripping for Automated Testing of Android Applications, *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, ACM, pp. 258–261 (2012).

[9] Azim, T. and Neamtiu, I.: Targeted and Depth-first Exploration for Systematic Testing of Android Apps, *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, ACM (2013).

[10] Choi, W., Necula, G. and Sen, K.: Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning, *SIGPLAN Not.*, Vol. 48, No. 10, pp. 623–640 (2013).

[11] Hao, S., Liu, B., Nath, S., Halfond, W. G. and Govindan, R.: PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps, *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, ACM, pp. 204–217 (2014).

[12] Deng, L., Mirzaei, N., Ammann, P. and Jeff, O.: Towards Mutation Analysis of Android Apps, *MUTATION 2015 (ICST 2015 Workshop)* (2015).

[13] Yang, S., Zhang, H., Wu, H., Wang, Y., Yan, D. and Rountev, A.: Static Window Transition Graphs for Android, *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE2015* (2015).

[14] Offutt, J.: muJava, `https://cs.gmu.edu/~offutt/mujava/`.