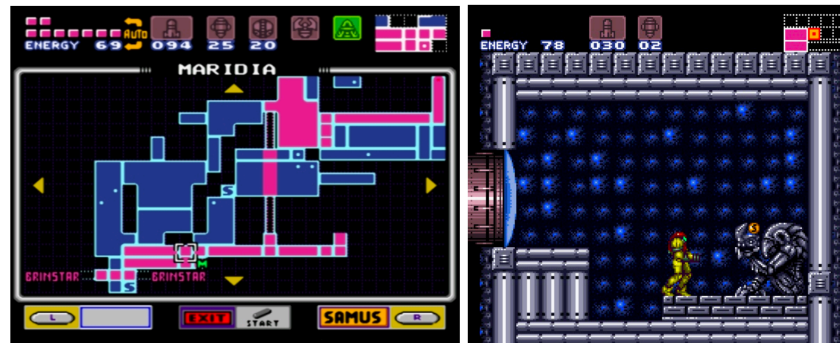


Time-travel Testing of Android Apps

Zhen Dong, Marcel Böhme, Lucia Cojocaru, Abhik
Roychoudhury

*Proceedings of the ACM/IEEE 42nd International Conference on
Software Engineering, June 2020, Pages 481–492*

Androidアプリのテスト



(a) Excerpt of the map of Maridia where pink marks explored rooms.

(b) Samus discovering the Spazer weapon

Figure 1: Super Metroid on an Android Emulator

- Androidアプリのテストはスーパーメトロイドというゲームを例えとして：
 - スーパーメトロイドは大きなマップでプレイされる
 - マップは、任意の順序で探索できるたくさんの部屋で構成されている
 - コントローラーのボタンを押すと、主人公のサムスがある部屋から次の部屋に移動し、秘密を見つけて敵と戦うことで力をつける
- マップ→全体のコード, 部屋→コード

背景

- Androidアプリのテストの重要性が増している
- Androidアプリのテストの問題：
 - 重要なところで保存できない
 - 間違ったときに時間を遡ることができない

目的

- Androidアプリ向けのタイムトラベルテストを提案する
- タイムトラベルテストフレームワークを実装し, 評価する
- 本研究のアプローチは, オープンソースとクローズドソースの両方のAndroidアプリのベンチマーク, およびGooglePlayの人気アプリトップ100で評価する
- タイムトラベルAndroidアプリテストツールTimeMachineをGithubで公開した
 - <https://github.com/DroidTest/TimeMachine>

用語

- タイムトラベルテスト
 - 過去に観察された任意の状態に戻る機能ため(過去の任意の状態に戻る機能)
- TimeMachine
 - 提案されたフレームワークの名前
 - 目的 : 状態とコードカバレッジを最大化すること

タイムトラベルフレームワーク

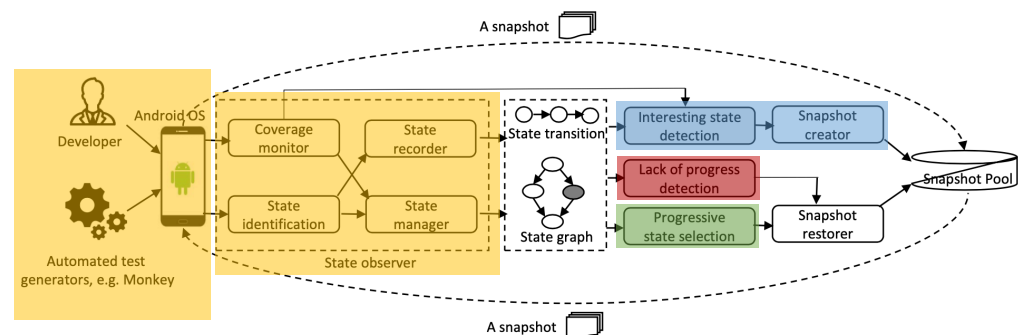


Figure 2: Time travel framework. Modules in grey are configurable, allowing users to adjust strategy according to scenarios.

1. Androidアプリは, 人間の開発者または自動テストジェネレーターで起動できる
2. アプリが操作されると, 状態観測モジュールは状態遷移を記録し, コードカバレッジの変化を観察する
3. 定義された基準を満たす状態は「interesting」としてマークされ, シミュレートされたAndroidデバイス全体のスナップショットを撮ることによって保存される
4. 「lack of progress」という状態が検出されると, フレームワークは現在の実行を終了し, 以前に記録された状態の中で最も進歩的なものを選択して復元する
5. 進歩的な状態に戻ると, 新しいイベントが起動される

タイムトラベルフレームワーク

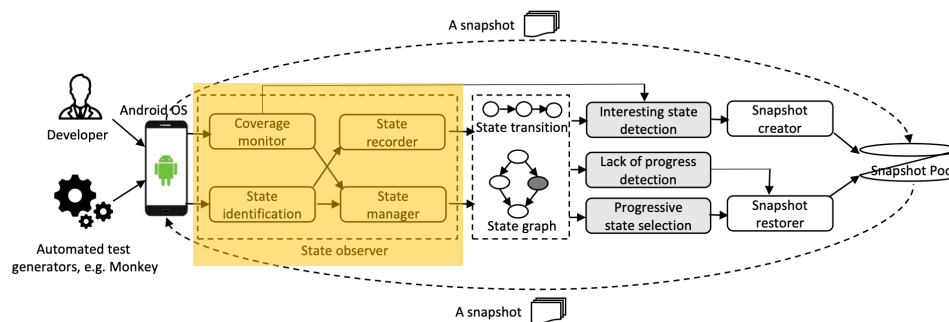


Figure 2: Time travel framework. Modules in grey are configurable, allowing users to adjust strategy according to scenarios.

1) 状態観測

• 状態識別

- Androidアプリのプログラム状態は, ウィジェット階層ツリーとして表される
- 状態は, ウィジェット階層ツリー上でハッシュを計算することによって一意に識別される

• 状態の保存と復元

- スナップショットに保持される情報：
 - 関連ファイル
 - データベース
 - サードパーティライブラリ
 - センサー
- スナップショットを復元することで状態を完全に再開できる

タイムトラベルフレームワーク

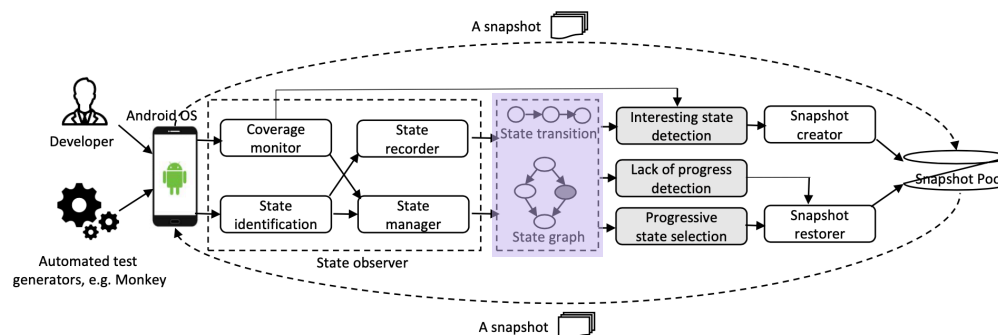


Figure 2: Time travel framework. Modules in grey are configurable, allowing users to adjust strategy according to scenarios.

2) 状態レベルのフィードバックの収集

- 有向グラフを使用して状態遷移を表す
 - ノードは検出された状態を示し, エッジは状態遷移を表す
- 各ノードは, 状態に関するいくつかの情報を維持する
 - スナップショットがあるかどうか (スナップショットのある状態のみを復元できる)
 - 訪問された頻度
 - 復元された頻度など
- この情報は, 状態がどの程度テストされているかを評価し, 実行をガイドするために, テストツールまたは人間のテスターに提供できる

タイムトラベルフレームワークs

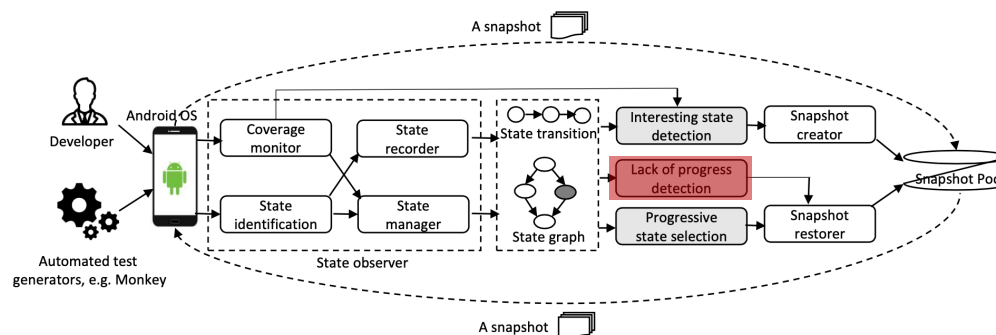


Figure 2: Time travel framework. Modules in grey are configurable, allowing users to adjust strategy according to scenarios.

4) 「lack of progress」 (進歩の欠如)を特定する

- テストプロセスは、かなり長い間、新しいプログラムの動作を発見することなく、進行しないままである可能性がある

タイムトラベルフレームワーク

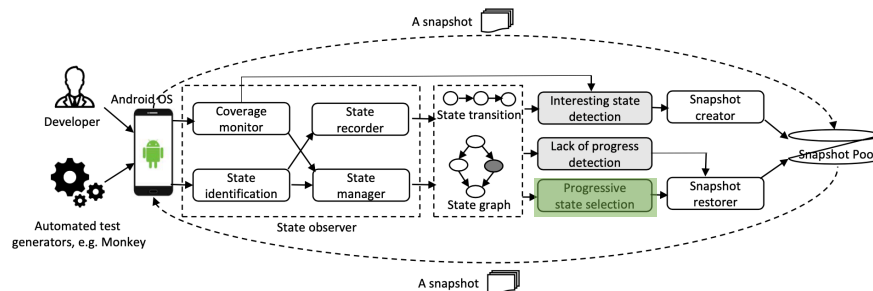


Figure 2: Time travel framework. Modules in grey are configurable, allowing users to adjust strategy according to scenarios.

Algorithm 3: Selecting the next state
(SELECTFITTESTSTATE)

Input: Path length k

```

1: procedure SELECTFITTESTSTATE( $states, stateGraph$ ) {
2:    $bestFitness \leftarrow 0$ 
3:   for each  $state$  in  $states$  do
4:      $stateFitness \leftarrow 0$ 
5:      $paths \leftarrow$  all paths in  $stateGraph$  of length  $k$  from  $state$ 
6:     for each  $path$  in  $paths$  do
7:       for each Node  $s$  in  $path$  do
8:          $stateFitness \leftarrow stateFitness + f(s)$  // see Eq. (1)
9:       end for
10:    end for
11:     $stateFitness \leftarrow \frac{stateFitness}{|paths|}$ 
12:    if  $stateFitness > bestFitness$  then
13:       $bestState = state$ 
14:       $bestFitness = stateFitness$ 
15:    end if
16:  end for
17:  return  $bestState$ 
18: }
```

5) 進歩的な状態の選択

- Monkeyが「isStuck」になった場合, 各状態に適合度を割り当てて, 新しいプログラムの動作をトリガーする可能性を評価する
- アルゴリズム3は、タイムトラベルのための最も進歩的な状態を選択するプロセスの概要を示す

TimeMachineのアルゴリズム

Algorithm 1: Time-travel testing (TimeMachine).

Input: Android *App*, Sequence generator *Monkey*

```
1: State curState ← LAUNCH(App)
2: Save VM snapshot of curState
3: Interesting states states ← {curState}
4: State Transition Graph stateGraph ← INITGRAPH(curState)
5: for each Event e in Monkey.GENERATEEVENT() do
6:   if timeout reached then break; end if
7:   prevState ← curState
8:   curState ← EXECUTEVENT(App, e)
9:   stateGraph ← UPDATEGRAPH(prevState, curState)
10:  if ISINTERESTING(curState, stateGraph) then
11:    Save VM snapshot of curState
12:    states ← states ∪ {curState}
13:  end if
14:  if ISSTUCK(curState, stateGraph) then
15:    curState ← SELECTFITTESTSTATE(states, stateGraph)
16:    Restore VM snapshot of curState
17:  end if
18: end for
```

Output: State Transition Graph *stateGraph*

- 1～4行目
 - TimeMachineは, 初期状態のスナップショットから始まる
- 5～9行目
 - Monkeyが生成するイベントごとに, 新しい状態が計算され, 状態遷移グラフが更新される
- 10～13行目
 - 状態が「Interesting」の場合, 仮想マシン (Virtual Machine, VM) のスナップショットが作成され, その状態に関連付けられる
- 14～17行目
 - Monkeyが「isStuck」や進歩が行われなくなった場合, TimeMachineは最も進歩的な状態を見つけ(selectFittestState), 関連するVMスナップショットを復元する
- 5～18行目
 - それ以外の場合は, 新しいイベントが生成され, ループが新たに開始される

アプローチ

- AndroidテストツールMonkeyにTimeMachineを追加し, タイムトラベルテストを実装した
- TimeMachineを評価する
 - RQ1: Monkey
 - RQ2: Monkey, StoaとSapienz
 - RQ3: IndustrialApps のベンチマークとGoogle Playの上位100個アプリ

リサーチクエスチョン

RQ1 :

- より多くのコードカバレッジを達成し, より多くのクラッシュを見つけるという点で, タイムトラベル手法はどの程度効果的であるか? TimeMachineとベースラインツールの比較

RQ2 :

- タイムトラベルテスト(TimeMachine)は, 達成されたコードカバレッジと検出されたクラッシュに関して, 既存の手法とどのように比較されるか?

RQ3 :

- タイムトラベルテスト(TimeMachine) は, 産業用アプリやGoogle Playのトップ100アプリなど, より大規模なアプリでどのように実行されるか?

RQ1：より多くのコードカバレッジを達成し, より多くのクラッシュを見つけるという点で, タイムトラベル手法はどの程度効果的であるか? TimeMachineとベースラインツールの比較

- TimeMachineとAndroTestで利用されているベースラインツールを評価する
- タイムトラベル手法を使用して, 達成されたコードカバレッジと検出された障害がどのように改善されるかを調査した
- MS：Stoatのシステムレベルのイベントジェネレーターで拡張されたMonkeyをベースライン
- MR：Monkeyの別のバリエーションを実装する

RQ1：より多くのコードカバレッジを達成し, より多くのクラッシュを見つけるという点で, タイムトラベル手法はどの程度効果的であるか? TimeMachineとベースラインツールの比較

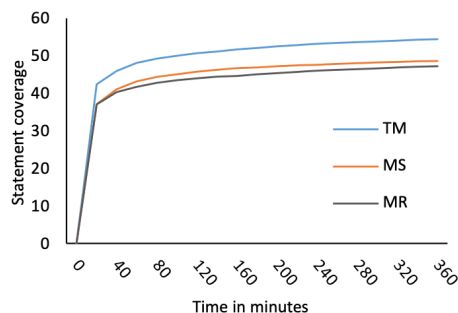


Figure 4: Progressive statement coverage for TimeMachine (TM) and baseline tools on 68 benchmark apps. MS indicates Monkey extended with Stoa's system-level generator and MR indicates Monkey with the ability to restart from scratch when lack of progress is detected.

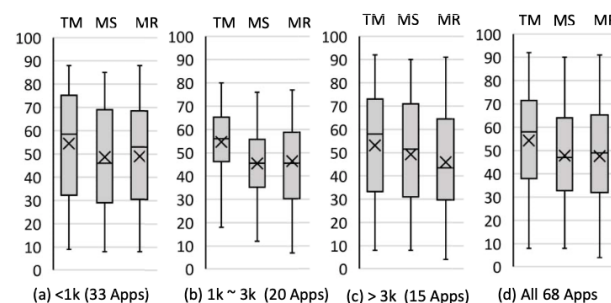


Figure 5: Statement coverage achieved by TimeMachine (TM), MS and MR.

- 図4は, 68個のアプリすべてについて実行時間にわたって達成されたコードカバレッジを示している
- 図5は, アプリのサイズごとに各ツールのステートメントカバレッジ結果を示している
 - 「x」は平均を示す
- 図4と図5は, TimeMachineが短時間でより多くのコードをカバーし, MRと比較してステートメントカバレッジを大幅に改善することを示している

RQ1 : より多くのコードカバレッジを達成し, より多くのクラッシュを見つけるという点で, タイムトラベル手法はどの程度効果的であるか? TimeMachineとベースラインツールの比較

手法	ステートメントカバレッジ (%)	クラッシュ (個)
TimeMachine	54	199
MS	47	115
MR	47	45

1.15倍

1.15倍

1.73倍

4.4倍

RQ2 : タイムトラベルテスト(TimeMachine)は, 達成されたコードカバレッジと検出されたクラッシュに関して, 既存の手法とどのように比較されるか?

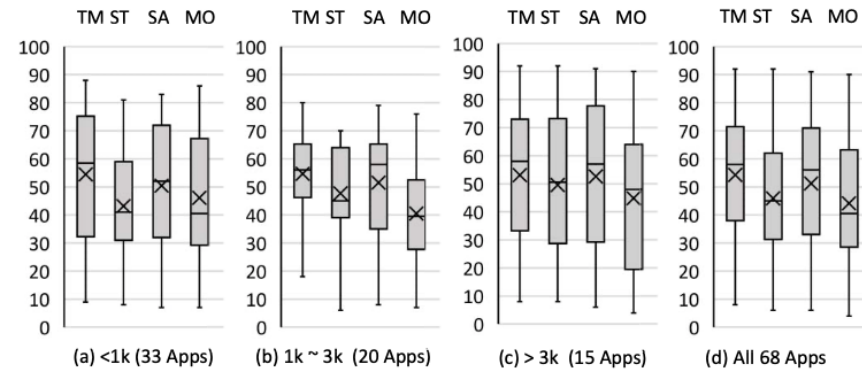


Figure 6: Statement coverage achieved by TimeMachine (TM), Stoa (ST), Sapienz (SA) and Monkey (MO).

- ステートメントカバレッジの平均：
 - TimeMachine (54%)
 - Sapienz (51%)
 - Stoa (45%)
 - Monkey (44%)
- 図6は, TimeMachineが4つのアプリサイズグループで一番ステートメントカバレッジが高い

RQ2 : タイムトラベルテスト(TimeMachine)は, 達成されたコードカバレッジと検出されたクラッシュに関して, 既存の手法とどのように比較されるか?

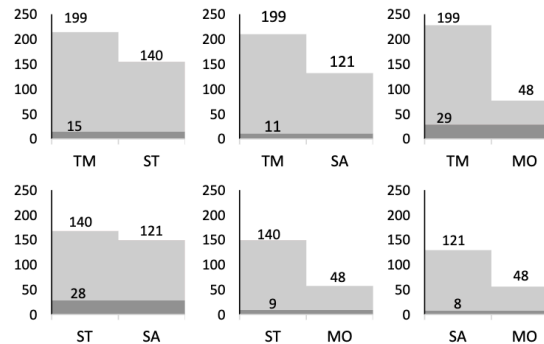


Figure 7: Comparison of total number of unique crashes for AndroTest apps. The dark grey areas indicate the proportion of crashes found by both techniques.

- 検出されたクラッシュは 10%未満の重なりがある
 - TimeMachineとStoat
 - TimeMachineとSapienz
- Monkeyとの重なりはかなり高い
- この分析は, TimeMachineを他のAndroidテスト手法と併用して, より多くのコードをカバーし, より多くのクラッシュを発見できることを示している

RQ3：タイムトラベルテスト(TimeMachine)は, 産業用アプリやGoogle Playのトップ100アプリなど, より大規模なアプリでどのように実行されるか?

- 表2は, 37のクローズドソースベンチマークアプリの結果を示している

- メソッドカバレッジ

- TimeMachine:19%
- MS: 17%
- MR: 15%

- 検出されたクラッシュの数

- TimeMachine:281
- MS:183 (1.5倍)
- MR:15(18.7倍)

Table 2: Results from 37 closed-source industrial apps.

Subject Name	#Method	%Coverage					#Crashes					#State TM
		TM	ST	MO	MS	MR	TM	ST	MO	MS	MR	
AutoScout24	49772	34	25	29	31	29	18	0	1	12	0	915
Best Hairstyles	28227	14	20	13	15	14	1	0	0	1	0	34
Crackle	48702	22	9	19	22	22	21	0	8	10	8	905
Duolingo	46382	26	13	22	23	22	12	0	0	8	0	384
ES File Explorer	47273	28	15	18	24	21	9	0	0	8	0	594
Evernote	45880	11	10	7	11	8	0	0	0	0	0	45
Excel	48849	19	9	14	14	14	2	0	0	0	0	201
Filters For Selfie	17145	9	13	8	9	8	0	0	0	0	0	43
Flipboard	41563	30	17	24	28	25	0	0	0	0	0	308
Floor Plan Creator	23303	29	30	23	26	26	0	0	0	0	0	394
Fox News	42569	28	13	21	20	17	5	0	1	4	0	635
G.P. Newsstand	50225	7	6	5	6	5	0	0	0	0	0	14
GO Launcher Z	45751	13	9	11	11	12	0	0	0	0	0	81
GoodRx	48222	24	19	21	22	21	17	0	0	11	0	468
ibisPaint X	47721	16	16	10	12	12	0	1	0	0	1	655
LINE Camera	47295	17	15	14	15	15	22	1	1	19	1	413
Marvel Comics	43578	18	18	15	17	15	0	0	0	0	0	133
Match	50436	15	11	11	15	11	0	0	0	0	0	106
Merriam-Webster	50436	25	21	18	24	23	6	0	2	3	2	1018
Mirror	36662	8	8	8	8	8	0	0	0	0	0	23
My baby Piano	20975	7	7	7	7	7	0	0	0	0	0	4
OfficeSuite	45876	17	11	16	16	16	3	0	0	1	0	479
OneNote	50100	11	11	11	10	11	23	0	4	9	2	181
Pinterest	46071	21	10	16	15	8	0	0	0	0	0	382
Quizlet	48369	33	22	30	33	31	23	0	0	17	0	548
Singing	46521	10	6	5	8	6	14	0	0	12	0	77
Speedometer	47773	16	11	13	13	15	0	0	0	0	0	51
Spotify	44556	13	14	10	11	10	0	0	0	0	0	36
TripAdvisor	46617	26	22	21	23	22	1	1	0	0	0	1279
trivago	50879	16	9	8	14	10	7	0	0	6	0	139
WatchESPN	43639	26	22	23	25	24	0	0	0	0	0	395
Wattpad	44069	25	14	13	22	13	0	0	0	0	0	327
WEBTOON	47465	21	17	13	19	16	12	0	0	8	1	487
Wish	48207	16	15	17	15	14	4	0	0	4	0	55
Word	49959	16	9	14	14	14	0	0	0	0	0	146
Yelp	46903	24	16	17	20	17	69	0	0	37	0	395
Zedge	46799	24	23	22	23	21	12	0	3	13	0	911
Ave/Sum	44182	19	14	15	17	15	281	3	20	183	15	358

RQ3：タイムトラベルテスト(TimeMachine)は, 産業用アプリやGoogle Playのトップ100アプリなど, より大規模なアプリでどのように実行されるか?

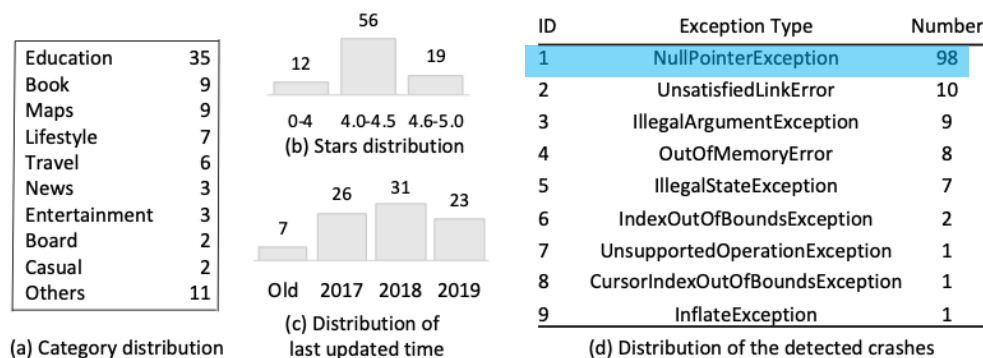


Figure 8: Statistics of tested 87 apps from Google Play.

- Google Playの上位100個アプリのうち, 87個のアプリのテストに成功した
 - 残りの13個のアプリは, 自己保護メカニズムのためにクラッシュし続けた
- 87個のアプリで, 137個の一意的なクラッシュが見つかった
- 検出された137のクラッシュは, 図8に示す9種類の例外が原因で発生した
 - 最も多いタイプはNullPointerException

結論

- タイムトラベルテストという手法を提案した
- TimeMachine :
 - AndroTestベンチマークで1.15倍のステートメントカバレッジを達成し, 1.7倍のクラッシュを発見した
 - コードカバレッジが平均的に高い値を達成し, 合計で最も多くのクラッシュが見つかった
 - 大規模な産業用アプリでも最先端の手法を上回る

所感

- Androidアプリテストの説明方が面白い
- Androidアプリのことも勉強になった

補足資料

- GitHubから得られた図

