GERADOR AUTOMATIZADO DE CASOS DE TESTE NA PLATAFORMA *ANDROID* BASEADO EM COMPUTAÇÃO EVOLUTIVA

Juscelino Barbosa da Silva Neto[1]
Filipe Grzelak Ribeiro Reis Antonelli[2]
Eduardo Noronha de Andrade Freitas[3]

[1]Instituto Federal de Educação, Ciência e Tecnologia de Goiás/Goiânia/Engenharia de Controle e Automação - PIBITI, juscelino.b@academico.ifg.edu.br
[2]Instituto Federal de Educação, Ciência e Tecnologia de Goiás/Goiânia/Engenharia de Controle e Automação - PIBITI, filipe.ifeca@gmail.com
[3]Instituto Federal de Educação, Ciência e Tecnologia de Goiás/Goiânia/DPAA IV, eduardonaf@gmail.com

Resumo

Com a popularização do sistema Android e a quantidade significativa de aplicativos disponíveis para uso, na Play Store, a possibilidade de haver falhas nos mesmos se tornou o principal mal a ser sanado, uma vez que estão envolvidos no processo, em boa parte das vezes, fatores de alta relevância, na esfera social, tais como: empresas de grande influência tecnológica, meios de comunicação multiplataforma, dentre outros. Com isso, surge a necessidade de um controle de qualidade eficiente e objetivo, visando encontrar a maior quantidade de erros possível. Como os testes eram, antes, realizados mecanicamente, por um testador físico, era necessário um tempo considerável, gerando também, mais gastos. No que compete à comunidade de desenvolvedores, foram elaboradas técnicas de geração de casos de testes UI (User Interface), de maneira mais otimizada e objetiva, uma vez que a participação humana no processo diminui, proporcionando, assim, a capacidade de uma geração de casos de testes mais rápida e com menos custos financeiros. Visto isso, foi feito o estudo das principais ferramentas voltadas à área de teste de *software*, afim de aprimorar os conhecimentos já existentes, na área. Com isso, foi proposta uma nova abordagem, nos casos de teste, na qual é analisada a relevância das *Activities* de cada aplicativo. Isso tem potencial para torna-los mais objetivos, diminuindo o tempo gasto com o processo e possibilitando um melhor direcionamento das áreas do aplicativo a serem testadas. À vista disso, foram feitas planilhas, com dados experimentais, a cargo de embasamento para o desenvolvimento do novo critério, bem como um *script*, que armazena todas as *Activities* exercitadas, durante o processo de testes. Esse novo critério de teste acrescenta considerável relevância aos estudos existentes, visto que aprimora o processo que, por sua vez, tem fundamental importância, na garantia de qualidade dos *softwares* desenvolvidos.

Palavras-chave: *Android*, aplicativos, casos de teste, *Activities*, *script*.

INTRODUÇÃO

Atualmente, o sistema *Android*, desenvolvido pela Google, é o mais utilizado, no que diz respeito a dispositivos móveis. Visto isso, a presença considerável do sistema, no mercado *mobile*, é refletida pela alta quantidade de aplicativos (cerca de 3,6 milhões) disponível à também considerável quantidade de usuários *Android* existente (mais de 2 bilhões).

Dada a grande quantidade e variedade de dispositivos e aplicativos existentes, no mercado móvel, é conveniente que haja um controle de qualidade, no que diz respeito à integridade dos *softwares* a serem entregues aos usuários finais, pela comunidade de

desenvolvedores. Esse controle é feito através de técnicas estáticas e dinâmicas, aplicadas direta ou indiretamente, aos dispositivos.

Uma dessas técnicas consiste no teste de *software*, que pressupõe a execução da aplicação, analisando a consonância entre as saídas esperadas e as obtidas, via testes, a partir de entradas pré-definidas. Quanto ao estabelecimento das entradas utilizadas na atividade de teste, podem ser concebidas das seguintes maneiras: 1) manualmente, 2) de forma semi-automatizada e 3) automatizada.

A primeira, mais comum, utiliza-se de meios mecânicos, expondo a execução via *hardware* da aplicação. A segunda realiza ações de acordo com a situação, mesclando atos realizados fisicamente, com eventos gerados de maneira automatizada. E, por fim, a terceira, como o próprio nome sugere, é dotada de automação, ou seja, é capaz de gerar ações sem a participação humana.

O impacto da automatização de testes de *software* é demonstrado na considerável redução de gastos com "testadores" humanos, uma vez que, utilizando-se de ferramentas computacionais, torna-se possível realizar uma maior quantidade de testes, num tempo consideravelmente menor e com menos problemas, durante a execução dos testes.

Haja vista a notória importância da atividade de testes, para a plataforma *Android*, convém-se destacar a considerável necessidade de eficiência, no que diz respeito aos processos realizados. Como um dos fatores preponderantes para a garantia de efetividade dos testes de *software*, está o tempo gasto, pois, ao passo que se reduz o tempo gasto com cada teste, aumenta-se a quantidade dos mesmos.

Outro fator levado em conta, durante o processo de teste de *software*, é a quantidade de erros revelados, uma vez que, quanto mais áreas erróneas do código forem encontradas, melhor e mais ampla será a correção feita pelos desenvolvedores.

Aliada a esses dois fatores anteriores, pode-se citar a quantidade de código coberta pelo teste feito. Já que, quanto mais partes do código do aplicativo forem testadas, maior é a abrangência do processo, possibilitando, assim, a obtenção de maior confiança. Juntamente a isso, pode-se citar o aumento da possibilidade de se encontrarem erros, no aplicativo, uma vez que mais áreas serão testadas, pondo à mostra uma maior parte do código e testando mais integralmente o produto a ser entregue aos usuários do sistema.

INÍCIO DAS ATIVIDADES

Durante o período de revisão bibliográfica, foi analisado o *AndroTest*, que consiste num projeto que contém sete (7) ferramentas voltadas ao teste de *softwares*, na plataforma *Android*. São as seguintes: *Monkey*, *Acteve*, *Dynodroid*, *A3E*, *SwiftHand*, *GuiRipper* e *PUMA*. E, além do projeto em questão, foi estudada a ferramenta *Sapienz*, que se utilizava de conceitos de inteligência artificial para a realização de testes.

Percebeu-se que, dentre todas as heurísticas utilizadas, havia uma considerada mais simplória e com menos capacidade de exploração da aplicação a ser testada: *Monkey*. Por outro lado, notou-se a presença de diversas ferramentas que utilizavam métodos mais complexos de teste, tornando o processo de busca por erros mais objetiva. Como exemplo, pode ser citada a ferramenta *Sapienz*.

MONKEY

Consiste num programa que é executado, via linha de comando, seja num emulador, seja num dispositivo físico, que gera fluxos pseudoaleatórios de eventos de usuário, tais como cliques, toques e gestos, bem como eventos a nível de sistema operacional.

Os pontos analisados foram: Cobertura de código, revelação de erros e tempo gasto. Pelo fato de serem realizados testes randômicos, ou seja, aleatórios, as aplicações não foram testadas de maneira objetiva, sendo apenas gerados eventos sem correlação aparente. Quanto à revelação de erros, não se tem parâmetros gerais, para determinar, com precisão, a rapidez com a qual o *Monkey* os encontra, durante os testes, levando, muitas vezes, muito tempo, para encontrar um erro que pode ser ou não relevante ao funcionamento geral da aplicação testada.

```
jhuscelino@Under:~/PIBITI$ adb shell
herolte:/ $ monkey -v 20
  bash arg: -v
  bash arg: 20
args: [-v, 20]
 arg: "-v"
 arg: "20"
:Monkey: seed=1534715476648 count=20
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//   0: 15.0%
//   1: 10.0%
//   2: 2.0%
//   3: 15.0%
//   4: -0.0%
//   5: -0.0%
//   6: 25.0%
//   7: 15.0%
//   8: 2.0%
//   9: 2.0%
//   10: 1.0%
//   11: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;component=com.whatsapp/.Main;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.whatsapp/.Main } in package com.whatsapp
:Sending Touch (ACTION_DOWN): 0:(669.0,677.0)
:Sending Touch (ACTION_UP): 0:(687.8442,687.277)
    // Allowing start of Intent { cmp=com.whatsapp/.HomeActivity } in package com.whatsapp
    // activityResuming(com.whatsapp)
:Sending Flip keyboardOpen=false
Got IOException performing flipjava.io.IOException: write failed: EINVAL (Invalid argument)
    // Injection Failed
:Sending Trackball (ACTION_MOVE): 0:(3.0,-1.0)
Events injected: 20
:Sending rotation degree=0, persist=false
:Dropped: keys=0 pointers=0 trackballs=0 flips=1 rotations=0
## Network stats: elapsed time=2614ms (0ms mobile, 0ms wifi, 2614ms not connected)
// Monkey finished
herolte:/ $
```

Figura 1: Execução da ferramenta Monkey.

SAPIENZ

Quanto a essa ferramenta, convém dizer que a mesma envolve conceitos relacionados a inteligência artificial, como, por exemplo, algoritmos genéticos. Em detrimento da Monkey, que realiza os testes de maneira randômica, a Sapienz passa a aprender acerca da aplicação que está sendo testada, após cada caso de testes.

Quanto aos pontos analisados nos estudos (Cobertura de código, revelação de erros e tempo gasto), a ferramenta em questão demonstra maior eficácia, visto que, pelo fato da mesma realizar testes gradativamente mais inteligentes, durante a execução dos casos de teste, são explorados pontos de maior relevância e com maior potencial para que sejam encontrados erros, em um tempo menor que o gasto pelas ferramentas anteriormente citadas.

DESCOBERTA DE UMA POSSÍVEL NOVA ABORDAGEM

Quando um aplicativo é concebido pela comunidade de desenvolvimento, o mesmo é fruto de alguma necessidade existente no contexto vivido pelo solicitante. Ou seja, cada nova aplicação tem, em si, o chamado "valor de negócio", que consiste em tudo que determina a saúde e o bem-estar de uma empresa, a longo prazo.

A cargo de esclarecimento e exemplificação, convêm serem citados os aplicativos voltados à realização de operações bancarias. Há determinadas partes que demonstram maior relevância, para a função para a qual o aplicativo foi concebido, do que outras. No caso dos aplicativos de bancos, as Activities (janelas) responsáveis pela realização de transferências monetárias, extrato bancário e pagamento de títulos se mostram mais relevantes do que as Activities que expõem informações técnicas sobre o aplicativo, por exemplo.

Nas atividades de teste de *software*, são abordados três métricas, já anteriormente citadas: Cobertura de código, revelação de erros e tempo gasto. Haja vista que a maioria dos programas responsáveis pela atividade de testes mensuram sua eficácia e eficiência, baseando-se nesses pilares, foi notado que um quesito importante para o sucesso da permanência do aplicativo no mercado *mobile* não foi abordado: As chamadas *Activities*.

IMPLICAÇÕES DA NOVA ABORDAGEM

Com o advento da nova abordagem, para a geração dos casos de teste, os mesmos passarão a ser mais objetivos, haja vista que, pelo fato de ser atribuída maior preferência de teste às *Activities* mais relevantes ao "valor de negócio", o processo de teste passará a tender majoritariamente àquela área de código correspondente à parte do aplicativo considerada mais relevante. Com isso, os testes passarão a ser menos inconclusivos, tendo em vista a objetividade adquirida.

Durante as pesquisas, foram confeccionadas planilhas, com resultados de análises manuais, no que diz respeito à importância das *Activities* de quatro aplicativos, levando em conta os seguintes critérios:

- Quantidade de componentes de tela
- Participação direta nas atividades

Os aplicativos testados foram: *Calendar*, *C:Geo*, *Daily Money* e *File Manager*. O sistema de notas vai de 1 (menor importância) a 5 (extrema importância).

A planilhas a seguir (Figura 2, 3, 4 e 5) contêm todas as *Activities* pertencentes ao aplicativos *Calendar*, *C:Geo*, *Daily Money* e *File Manager*, bem como os atributos de cada uma (Texto, por exemplo). Para cada *Activity*, foi atribuída uma nota, de acordo com a importância da mesma para a função principal de cada aplicativo.

	Main	1) Configurações	1.1) Personalizar cores	1.2) Gerenciar tipos de eventos	1.3) Som do lembrete	Sobre
Calendário	5	5	3	3	4	1
Identificação (resource-id):	com.simplemobiletools.calendar:id/day_41	com.simplemobiletools.calendar:id/weekly_view_label	com.simplemobiletools.calendar:id/customization_theme	com.simplemobiletools.calendar:id/add_event_type	android:id/text1	com.simplemobiletools.calendar:id/about_invite
Texto:					One Drop	Convidar amigos

Figura 2: Planilha com notas estipuladas manualmente do aplicativo *Calendar*.

	Main	Ao Vivo	Por Perto	Arquivo	Pesquisar	Ir Para	Histórico	Definições (1)	1) Serviços (1.1)
C:Geo	5	5	4	3	5	5	2	3	2
resource-id:	android:id/ac	android:id/ac	android:id/ac	android:id/te	android:id/ac	android:id/ac	android:id/te	android:id/action_bar_title	
Text:	c:geo	Mapa ao vivo	Por perto	Arquivadas	Pesquisar	Destino	Histórico	c:geo Definições	

Figura 3: Planilha com notas estipuladas manualmente do aplicativo *C:Geo*.

	Main	Reports	Add Detail	Daily List	Weekly List	Monthly List	Yearly List	Accounts	Data	Preferences	Books	About
Daily Money	5	5	5	3	3	3	3	3	5	3	3	1
resource-id:	com.bottlework s.dailymoney :id/dt_grid	com.bottlework s.dailymoney :id/dt_label	com.bottlework s.dailymoney :id/ deteditor_ok	com.bottlework s.dailymoney :id/ detlist_mode	com.bottlework s.dailymoney :id/ detlist_mode	ks.dailymoney :id/ detlist_sum_as set	ks.dailymoney :id/ detlist_item_fi om	android:id/ title	ks.dailymoney :id/ datamain_crea te_default	android:id/ title	com.bottlework s.dailymoney :id/ bookmgnt_list	ks.dailymoney :id/ about_whatsne w
Text:		Cumulative Balance	Create					Income	Create default data(account)	Last Data Backup Time		
INDEX					3							
NAF					TRUE							

Figura 4: Planilha com notas estipuladas manualmente do aplicativo *Daily Money*.

	Main	Configuraçõe	Personalizar	Gerenciar Fa	Sobre			
File Manager	5	5	2	2	1			
resource-id	com.simplem	com.simplem	com.simplem	com.simplem	com.simplemobiletools.filemanager:id/add_favorite			
Text	Alarms	Personalizar	Tema	Pode adicionar pastas como favoritos, para um acesso mais rápido.				

Figura 5: Planilha com notas estipuladas manualmente do aplicativo *File Manager*.

TRABALHO FEITO (*SCRIPT*)

Conforme as Figuras 6, 7 e 8, foi feito um *script*, em *Python*, com o intuito de verificar quais as *Activities* eram abertas, durante a execução dos casos de teste, dando embasamento ao novo critério (importância das *Activities*). O *script* verifica a qual *package* (pacote) a *Activity* executada pertence (Figura 6), bem como o nome da *Activity* (Figura 7). Com isso, esses dados são armazenados num arquivo chamado *Results.txt* (Figura 8), que servirá de base para o cálculo da nota para os casos de teste, possibilitando, assim, o direcionamento das atividades de teste de *software*.

```
30 def get_package():
31         output = subprocess.Popen("adb shell dumpsys window windows | grep mCurrentFocus | cut -d '/' -f1 |
   rev | cut -d ' ' -f1 | rev", shell=True, stdout=subprocess.PIPE).communicate()[0]
32         output = str(output)
33         output = output.replace("\\n", "\n")
34         output = output.replace("b'","")
35         output = output.replace("'","")
36         print(output)
37         return output
```

Figura 6: Comando para obtenção do *package* (pacote).

```
21 def get_activity():
22         output = subprocess.Popen("adb shell dumpsys activity activities | grep mFocusedActivity | cut -d ' '
   -f 6 | grep -o '[^.]*$'", shell=True, stdout=subprocess.PIPE).communicate()[0]
23         output = str(output)
24         output = output.replace("\\n", "\t\t")
25         output = output.replace("b'","")
26         output = output.replace("'","")
27         print(output)
28         return output
```

Figura 7: Comando para obtenção da *Activity*.

```
39 def get_arquivo(activity,package):
40         with open("Results.txt", "a") as arq:
41                 arq.write(activity)
42                 arq.write(package)
43
44 def results_organize():
45         with open("Results.txt", "a") as arq:
46                 arq.write("ACTIVITIES\t\t")
47                 arq.write("PACKAGES\n\n")
```

Figura 8: Comando para armazenar os *packages* e *Activities* num arquivo "*Results.txt*".

CONCLUSÃO

A partir de tudo isso, nota-se a considerável importância do processo de teste de *software*, como verificador de qualidade das aplicações desenvolvidas. Com isso, visando aperfeiçoar as heurísticas já existentes, no mercado, torna-se conveniente fazer uso da nova abordagem, já citada anteriormente.

Tendo em vista os benefícios ocasionados no processo de teste, é plausível utilizar o novo critério, levando em conta a relevância das *Activities*. Com a introdução dessa nova métrica, os testes poderão se tornar mais objetivos e passar a gastar menos tempo, pelo fato de buscar testar majoritariamente as áreas de código consideradas mais importantes, para a função natural do aplicativo.

Através dos estudos feitos e dos conhecimentos adquiridos, durante todo o período de pesquisa, é possível afirmar que essa é uma potencial melhoria nas heurísticas já existentes, uma vez que utiliza um novo critério de testes, deixando os casos de teste mais precisos e assertivos, dentro de um conjunto de parâmetros pré-estabelecidos de avaliação de casos de teste.

REFERÊNCIAS

ANAND, Saswat, HARROLD, Mary Jean, NAIK, Mayur, YANG, Hongseok. **Acteve.** Disponível em : <https://code.google.com/archive/p/acteve/>. Acesso em: 16 de agosto de 2018.

Automatic Android App Explorer. Disponível em: <http://spruce.cs.ucr.edu/a3e/>. Acesso em: 16 de agosto de 2018.

CHOUDHARY, Shauvik Roy, GORLA, Alessandra, ORSO, Alessandro. **AndroTest - Automated Test Input Generation for Android: Are We Yet?** Disponível em: <http://bear.cc.gatech.edu/~shauvik/androtest/>. Acesso em: 16 de agosto de 2018.

CIRIACO, Douglas. **Pesquisa: Play Store aumentou 30% e App Store diminuiu 5% em 2017.** Disponível em: <https://www.tecmundo.com.br/software/128979-pesquisa-play-store-aumentou-30-app-store-diminuiu-5-2017.htm>. Acesso em: 16 de agosto de 2018.

Gui Ripper Wiki. Disponível em: <http://wpage.unina.it/ptramont/GUIRipperWiki.htm>. Acesso em: 16 de agosto de 2018.

LIMA, Mariana. **Aos 10 anos e com 2 bilhões de usuários, sistema *Android* mira emergentes.** Disponível em: <https://link.estadao.com.br/noticias/cultura-digital,aos-10-anos-e-com-2-bilhoes-de-usuarios-sistema-android-mira-emergentes,70002212394>. Acesso em: 16 de agosto de 2018.

SwiftHand: Automated Testing Tool for Android Applications. Disponível em: <https://github.com/wtchoi/SwiftHand>. Acesso em: 16 de agosto de 2018.

UI/Application Exerciser Monkey. Disponível em: <https://developer.android.com/studio/test/monkey>. Acesso em: 16 de agosto de 2018.

USC-NSL/PUMA: Programmable UI-Automation Framework for Dynamic App Analysis. Disponível em: <https://github.com/USC-NSL/PUMA>. Acesso em: 16 de agosto de 2018.