Automated Accessibility Testing of Mobile Apps

By Marcelo Medeiros Eler*, Jose Miguel Rojas§, Yan Ge†, and Gordon Fraser‡

Pesquisas na área de testes para mobile apps teve grandes avanços

eficiência
de energia

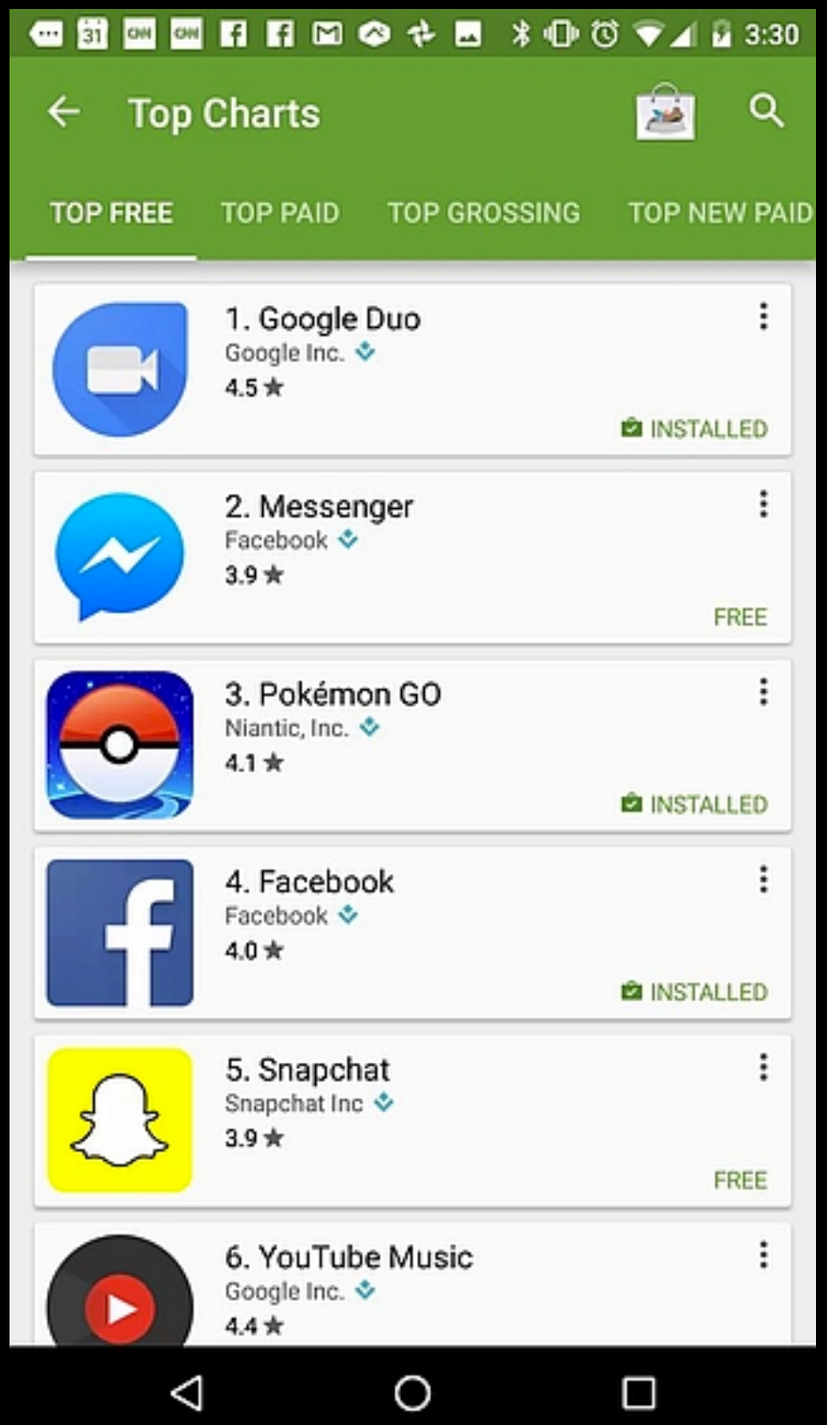
segurança
&
privacidade

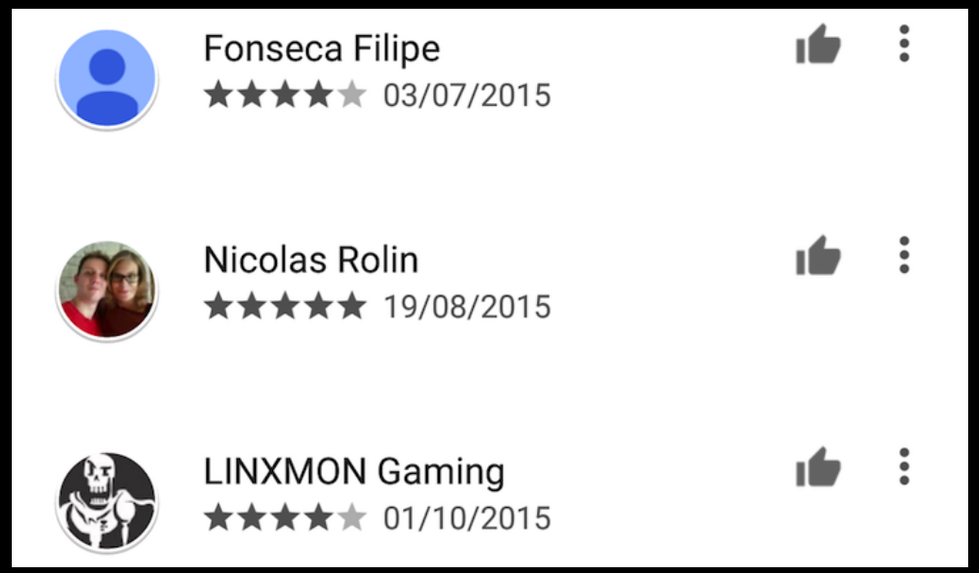
performance

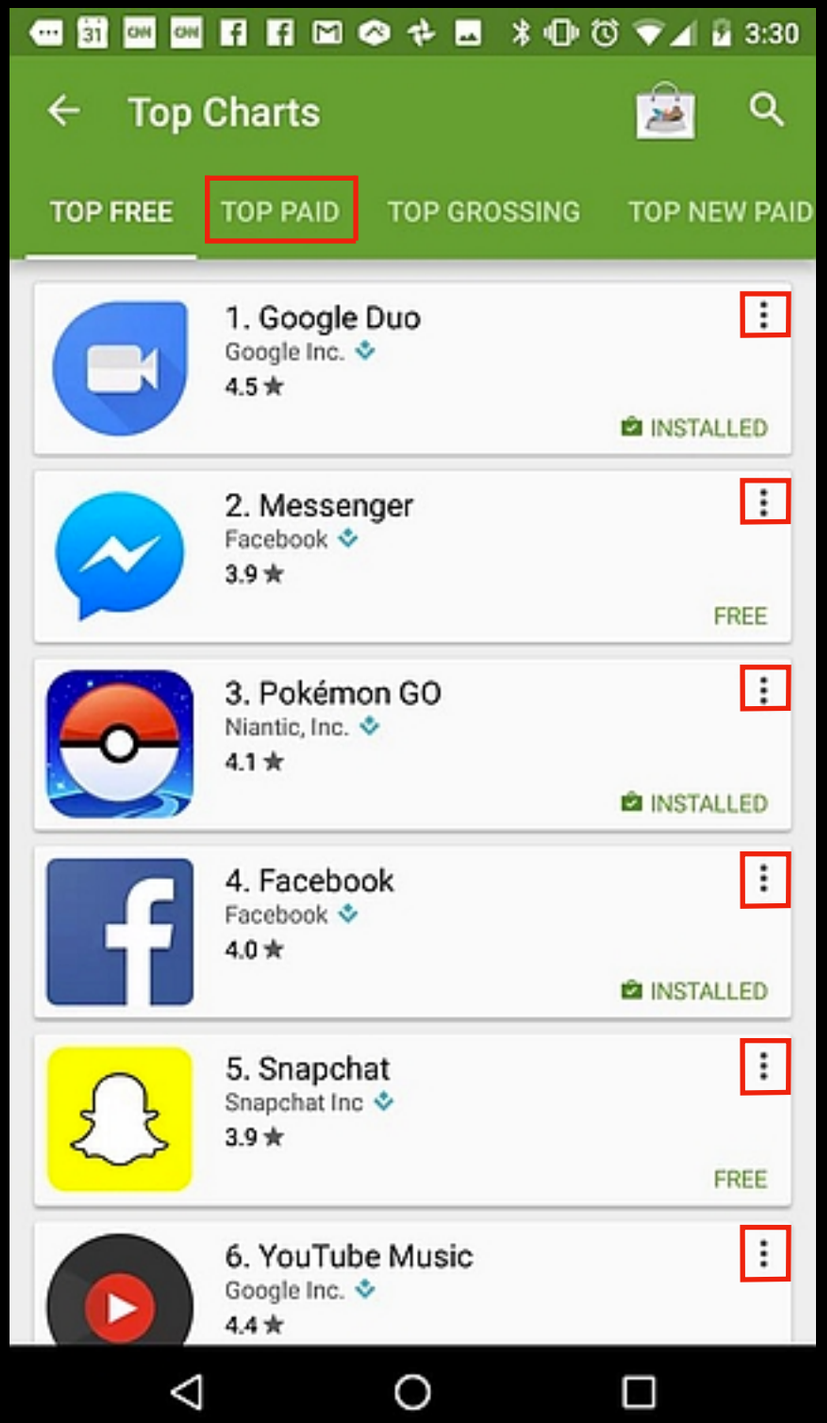
acessibilidade?

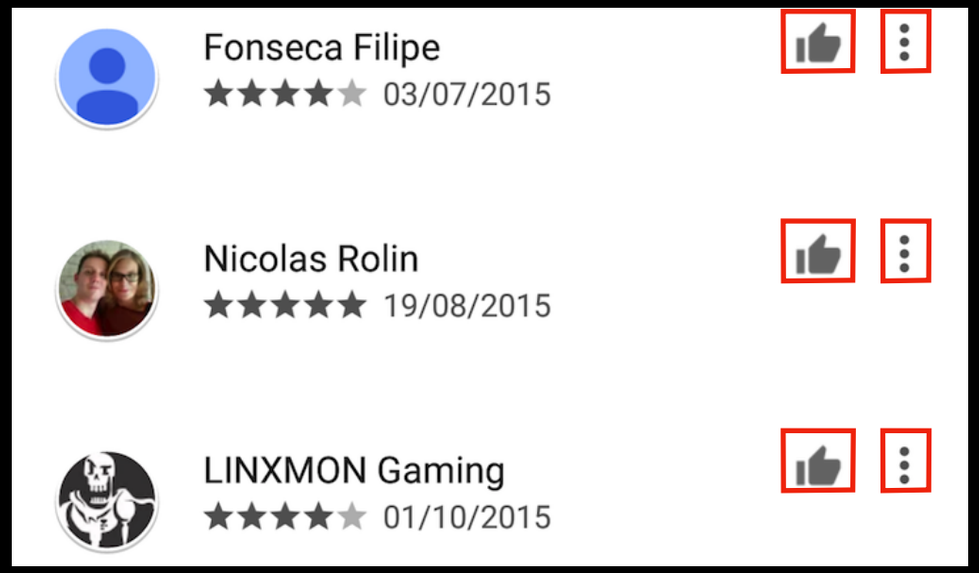
15% da população mundial possui algum tipo de deficiência e encara limitações na vida regularmente (incluindo o acesso a tecnologias de informação e comunicação)

World Bank. Main report


3:30
Top Charts
TOP FREE
TOP PAID
TOP GROSSING
TOP NEW PAID
1. Google Duo
Google Inc.
4.5
INSTALLED
2. Messenger
Facebook
3.9
FREE
3. Pokémon GO
Niantic, Inc.
4.1
INSTALLED
4. Facebook
Facebook
4.0
INSTALLED
5. Snapchat
Snapchat Inc
3.9
FREE
6. YouTube Music
Google Inc.
4.4


Fonseca Filipe
03/07/2015
Nicolas Rolin
19/08/2015
LINXMON Gaming
01/10/2015


3:30
Top Charts
TOP FREE
TOP PAID
TOP GROSSING
TOP NEW PAID
1. Google Duo
Google Inc.
4.5
INSTALLED
2. Messenger
Facebook
3.9
FREE
3. Pokémon GO
Niantic, Inc.
4.1
INSTALLED
4. Facebook
Facebook
4.0
INSTALLED
5. Snapchat
Snapchat Inc
3.9
FREE
6. YouTube Music
Google Inc.
4.4


Fonseca Filipe
03/07/2015
Nicolas Rolin
19/08/2015
LINXMON Gaming
01/10/2015

Make apps more accessible

Android apps should aim to be usable by everyone, including people with accessibility needs.

People with impaired vision, color blindness, impaired hearing, impaired dexterity, cognitive disabilities, and many other disabilities use Android devices to complete tasks in their day-to-day lives. When you develop apps with accessibility in mind, you make the user experience better, particularly for users with these and other accessibility needs.

This document presents guidelines for implementing key elements of accessibility so that everyone can use your app more easily. For more in-depth guidance on how to make your app more accessible, visit the principles for improving app accessibility page.

Increase text visibility

For each set of text within your app, the *color contrast* – or difference in perceived brightness between the color of the text and the color of the background behind the text – is recommended to be above a specific threshold. The exact threshold depends on the text's font size and whether the text appears in bold:

16:37 ··· 42%

acessibilidade

Scanner de acessibilidade
Google LLC
2,3 MB 4,6★

Accessibility Settings Shortcut
dogfooder
2,8 MB 4,2★

Guiaderodas acessibilidade
guiaderodas
9,7 MB 4,7★

CittaMobi Acessibilidade
CittaMobi
2,0 MB

APT "Acessibilidade Para Todos"
Guilherme Dorazzi
18 MB 4,9★

acessibilidade - caminhos acessíveis...
Valmir Marques
8,5 MB 4,0★

Detector de Cor
RamelTec
1,6 MB 4,3★

Simple Control
coolAce

“Acessibilidade mobile” refere-se a tornar websites e aplicativos mais acessíveis para pessoas com deficiências quando estes utilizam celulares e outros devices.

World Wide Web Consortium (W3C)

Tipos de testes de acessibilidade

manuais

automatizados

Testes Manuais

Google’s Accessibility Test Framework

- UIAutomatorViewer
- Acessibility Scanner

Testes Manuais

Explorar o app como o usuário o utilizará

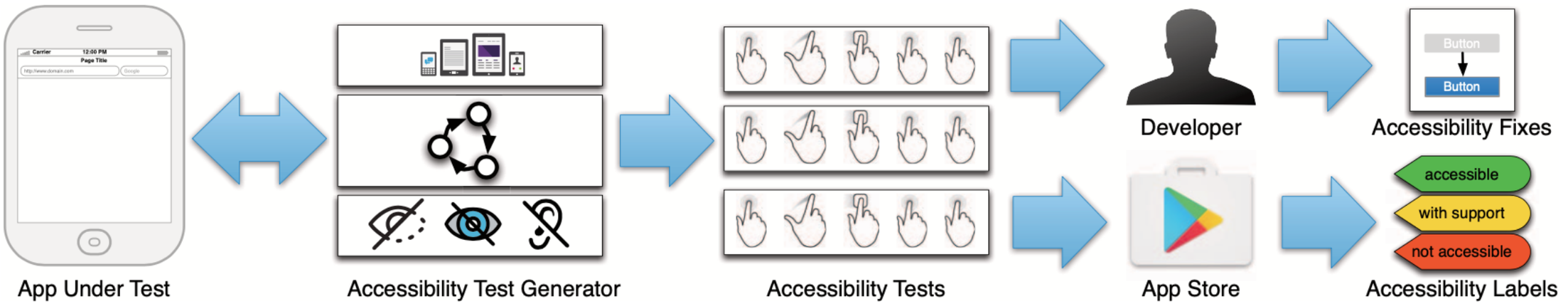
Testes Automatizados

- **Android Lint** - análise estática
- **Espresso** e **Roboletric** - análise dinâmica por testes de UI (testar acessibilidade não é o objetivo principal)

Frameworks de testes de UI que fazem **análise dinâmica** dos componentes para testar automaticamente requisitos de acessibilidade (por mais que não seja o objetivo principal delas)

A efetividade dessas ferramentas **depende diretamente da completude dos casos de teste** disponíveis para o app

Testes de acessibilidade automatizados


Carrier
12:00 PM
Page Title
Button
Button
accessible
with support
not accessible
App Under Test
Accessibility Test Generator
Accessibility Tests
Developer
App Store
Accessibility Fixes
Accessibility Labels

Propósito dos Testes

→ Guia para que os desenvolvedores melhorem a acessibilidade do app

→ Métrica de qualidade para lojas de apps

- acessível
- com suporte
- não acessível

MATE: The Mobile Accessibility Testing Tool

Características do MATE

→ Explora o app de forma **randômica** durante os testes

→ Ao invés de produzir eventos randômicos em locais randomicos, ele identifica todas as interações de usuário possíveis e aleatoriamente seleciona uma delas

Características do MATE

→ O app é resetado após um dado número de eventos

→ Dados de inputs são random também (já no formato que são requeridos)

Características do MATE

→ Também faz checagens de acessibilidade nos elementos da UI

→ No final, o MATE reporta o **número** de falhas únicas de cada tipo de checagem implementado, assim como um **relatório detalhado** de todas falhas de acessibilidade

Características do MATE

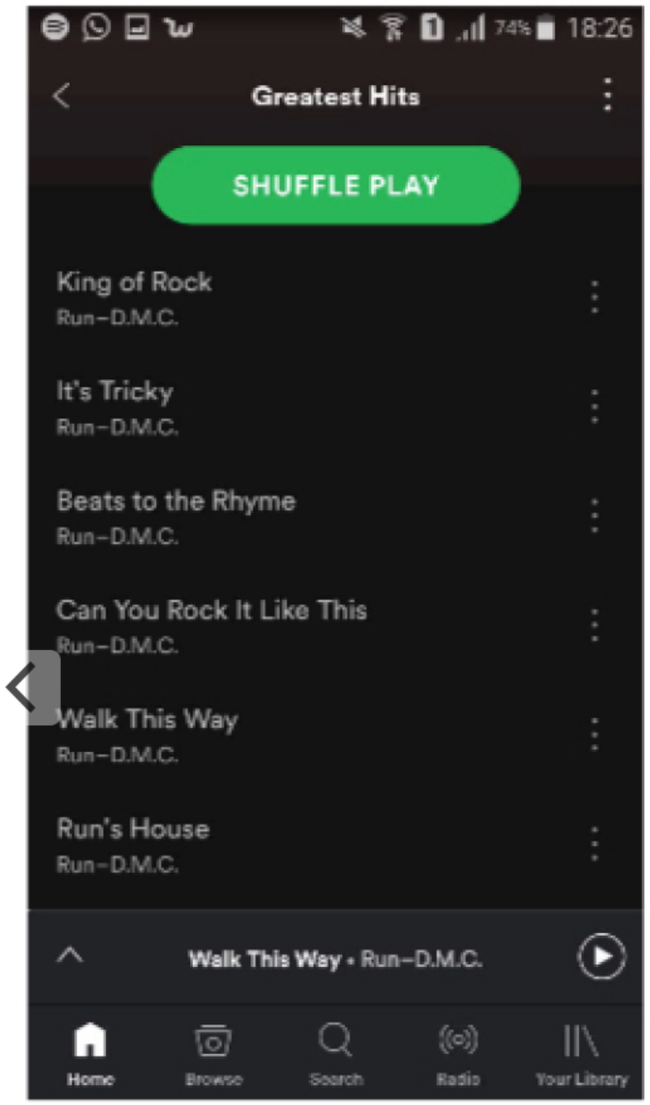
→ Roda no ambiente Android
→ Usa a **UIAutomator** framework para interagir com o app
→ Coleta informações na atividade **em nível de execução** em termos de informações disponíveis em cada view

Características do MATE

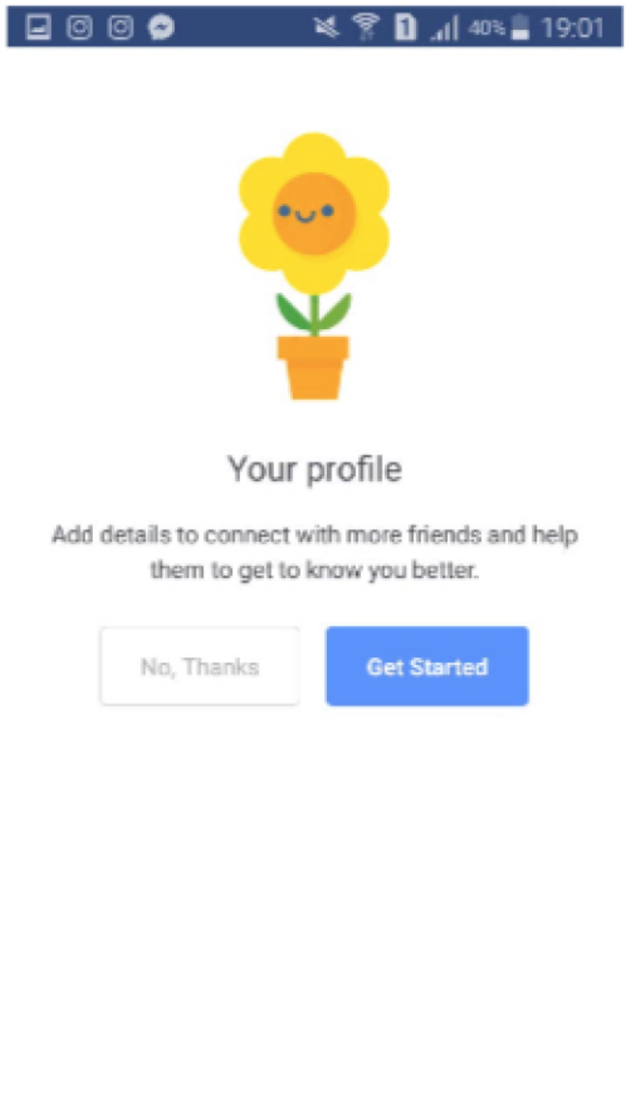
→ Não requer o source code sobre testes
→ Para utilizá-lo, apenas abra o app no device ou emulador e execute o MATE, que irá testar o app até atingir o critério de parada

Propriedades de Acessibilidade testadas no MATE

1. Texto pronunciável
2. Área de toque
3. Taxa de contraste
4. Limites clicáveis duplicados
5. Extensões clicáveis



(a) Spotify: The back button (<) has missing speakable text, and the too small ":" buttons all have the same duplicate text.



(b) Facebook: The "No, Thanks" button has low contrast.

Fig. 3: Examples of accessibility flaws checked by MATE.

Características do MATE

→ Também usa a framework de testes de acessibilidade da Google (reutiliza e adapta)

→ Otimização de checagem de acessibilidade com **abstração de estados**

Abstração de estados

→ O MATE utiliza uma estratégia baseada em **modelo**:

- Este é produzido automaticamente durante a exploração do app
- Representa as configurações de tela (estados) e transições entre telas
- Cada transição é causada por um evento ou interação do usuário

Abstração de estados

→ Durante exploração, MATE checa se um novo estado foi criado

- Se sim, atualiza o modelo

→ Checagens de acessibilidade só acontecem se um novo estado foi criado

→ O critério pra distinguir diferentes estados é importante

→ Modelo pode ser representado como um grafo direcionado

Avaliação da eficiência do MATE

Avaliação

- **RQ1**: O quão efetiva é a abstração de estados do MATE?
- **RQ2**: Como o MATE se compara a análises estatísticas do Android Lint?
- **RQ3**: Como o MATE se compara a frameworks de testes de acessibilidade dependentes de testes existentes?
- **RQ4**: Pode o MATE achar falhas de acessibilidade que não podem ser evitadas por features próprias do Android?

Teste para a Research Question 1:

- **Objetivo**: Detectar a perda de informações e falsos negativos por usar o sistema de abstração de dados do MATE.
- Teste feito com 10 aplicativos F-Droid aleatórios, número fixo de 1500 eventos, com 5 repetições de escolha.
- Outro teste consistindo de tempo fixo, mais real nas situações de empresas que estão fazendo testes.

Resultado da Research Question 1:

- Há o payoff de tempo-perda na detecção de falhas, como esperado.
- Qualquer perda é relevante dada a gravidade do tema (acessibilidade)
- Em tempo fixo o **MATE** se saiu muito bem graças a abstracao de estados.

Teste para a Research Question 2:

- Android LINT e o estado da arte atual para análise de acessibilidade e testes, por isso sua escolha.
- 50 aplicativos da F-Droid, mas a limitação do LINT de ser colocado junto a build limitou alguns apps que os autores não conseguiram dar build.
- Tempo fixo de 30 minutos com 10 repetições.

Resultado da Research Question 2:

- Ambos possuem pontos positivos, até por serem aplicativos que funcionam de maneira diferente. O LINT e estático e analisa source code.
- O MATE detecta em media mais erros de acessibilidade, graças à deteção dinâmica que detectam falhas de widgets criados em runtime.
- O LINT porém detecta falhas em source code, apesar de isso não ser muito comum nos aplicativos testados.

Teste para a Research Question 3:

- Foram comparados com Espresso e Roboletric:
  - 12 apps para o Espresso e 13 para o Roboletric.
- A limitação foi graças à necessidade de casos de testes existentes para rodar os softwares.
- 30 minutos e 10 repetições. Alguns apps não tinham configuração de acessibilidade então foi ligado manualmente pelos autores, o que pode ter afetado o resultado.

Resultado da Research Question 3:

- Alguns tipos de problema, especialmente os de tamanho de fonte e constraste são detectados pelo MATE e não pelos outros.
- Porém outros tipos de falhas são detectadas integralmente por qualquer software.
- Caso os apps tivessem os casos de testes mais robustos, como seria numa empresa, eles poderiam melhorar a performance dos concorrentes.

Teste para a Research Question 4:

- O próprio Android já tem a feature de melhorar as falhas de acessibilidade.
- O MATE foi testado em aplicativos com essa feature ligada.
- 50 apps, 30 minutos fixo e 10 repetições.

Resultado da Research Question 4:

- Principalmente contraste foram diminuídas as falhas.
- Porém em outros critérios, o MATE não deixou de detectar erros.

Dados

- Os dados encontrados em mais detalhes foram postos em tabelas, comparando com os programas previamente citados Lint, Espresso e RoboEletric em diversos critérios que foram pincelados na explicação Research Questions anteriormente.
 - A abundância de comparações deixa difícil de analisar caso a caso, mas em geral, o MATE se saiu bem melhor.
 - Vale ressaltar que os critérios foram escolhidos pelos autores e nao foram feitos testes com usuários para saber quais seriam mais relevantes ou qual foi a melhora qualitativa.

TABLE III: RQ2. Comparison of number of flaws found by MATE and Lint.

App	Activity Coverage		Overall		Size		Contrast		Duplicate Text		Missing Text		Editable Content		Click Bounds		Click Span		Clickable View	
	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE	Lint	MATE
arXiv mobile	-	0.8	5.0	30.8	-	9.8	-	14.4	-	0.0	5.0	4.3	-	0.0	-	2.3	-	0.0	0.0	-

Conclusões

Conclusões

→ Quanto mais apps se tornam importantes, mais **necessário** se torna acessibilidade para que não ocorra uma exclusão

- **Requisito**, não feature

→ Ferramentas de suporte aos desenvolvedores ainda são limitadas por análise **estática** (quando boa parte dos apps possuem natureza **dinâmica**)

→ Introduzimos o conceito de **automação de testes de acessibilidade**

- Geração automática de testes + análise dinâmica

Conclusões

→ Protótipo **MATE**: Ferramenta que testa propriedades de acessibilidade automaticamente

→ Supera todos os problemas de todas as abordagens atuais:

- Propriedades testadas em **runtime**
- Como gera testes, não precisa de uma base forte de testes pré-definida
- Experimentos mostram que supera problemas da checagem estatística do Android Lint e detecta mais problemas de acessibilidade que o Espresso ou Roboletric

Conclusões

→ O **MATE** serve apenas como uma prova de conceito de **automação de testes de acessibilidade**:

- **Estudo de usuário** - baseado em guidelines gerais de acessibilidade
- **Estudo de desenvolvimento** - intenção de avaliar o quão efetivo o MATE é para ajudar desenvolvedores reais a identificar e consertar problemas de acessibilidade
- **Propriedades de acessibilidade** - implementa algumas checagens de acessibilidade para deficiências **visual, física e motora**

Conclusões

→ O **MATE** serve apenas como uma prova de conceito de **automação de testes de acessibilidade**:

- **Geração de testes melhorada** - tem potencial de melhorar a medida que a tecnologia de geração de testes avança
- **Certificação de acessibilidade** - um dos objetivos é criar uma medida de certificação que apps são acessíveis

Perguntas?