Capítulo 1

Como testar a acessibilidade em soluções *mobile*

Anderson C. Garcia, Juliana M. L. Eusébio e Kamila R. H. Rodrigues
[1]Instituto de Ciências Matemáticas e de Computação (ICMC), USP
{andersongarcia,julianaleoncio}@usp.br, kamila.rios@icmc.usp.br

Abstract

Currently, native Android apps still present interaction barriers for users with disabilities, emphasizing the ongoing need to enhance the accessibility of these applications. Research indicates that developers have shown some awareness of the importance of accessibility, but knowledge on the subject is superficial, practical guidance is lacking, accessibility requirements are not well-specified, responsibilities are not well-defined, and accessibility is not prioritized by stakeholders. This chapter aims to address these challenges and empower development, design, and testing professionals to tackle the complexities of accessibility in mobile solutions, exploring approaches and tools for testing and improving accessibility.

Resumo

Atualmente, aplicativos Android nativos ainda apresentam barreiras que dificultam a interação dos usuários com deficiência, o que ressalta a necessidade contínua de melhorar a acessibilidade desses aplicativos. Pesquisas indicam que os desenvolvedores têm apresentado alguma consciência sobre a importância da acessibilidade, porém o conhecimento sobre o tema é superficial, faltam orientações práticas, requisitos de acessibilidade não são bem especificados, as responsabilidades não são bem definidas e a acessibilidade não é priorizada pelas partes interessadas. Este capítulo visa abordar esses desafios e capacitar profissionais de desenvolvimento, design e teste para enfrentar as complexidades da acessibilidade em soluções mobile, explorando abordagens e ferramentas para testar e melhorar a acessibilidade.

1.1. Introdução

A acessibilidade é um direito fundamental, defendido pela "Lei Brasileira de Inclusão" [1]. Acessibilidade significa eliminar barreiras para permitir o acesso de todas as pessoas, independentemente de deficiências ou outras limitações. Para cumprir essa missão, a acessibilidade é vital, especialmente no contexto das soluções computacionais.

Atualmente, aplicativos Android nativos ainda apresentam barreiras que dificultam a interação dos usuários, o que ressalta a necessidade contínua de melhorar a acessibilidade desses aplicativos. Existem instrumentos disponíveis, como guias, recomendações e ferramentas de testes, para auxiliar os desenvolvedores a eliminarem essas barreiras em seus aplicativos. No entanto, se observa que, apesar da disponibilidade dessas soluções, muitos aplicativos continuam apresentando problemas recorrentes [2, 3], o que indica que essas ferramentas podem não estar sendo adequadamente aplicadas.

Pesquisas têm buscado entender a razão de designers e desenvolvedores não implementarem produtos acessíveis [4, 5, 6, 7]. Embora considerem diferentes contextos de desenvolvimento, como *Web*, *mobile* e desenvolvimento ágil de software, essas pesquisas convergem em suas conclusões. Entre os argumentos, é possível citar: a) desenvolvedores têm apresentado alguma consciência sobre a importância da acessibilidade, porém o conhecimento sobre o tema é superficial, b) faltam orientações práticas, c) requisitos de acessibilidade não são bem especificados, d) as responsabilidades não são bem definidas e e) a acessibilidade não é priorizada pelas partes interessadas. Entre as possíveis soluções, apontadas nos estudos para esses problemas, destacam-se: a) oferecer cursos e treinamentos para ampliar o conhecimento dos desenvolvedores sobre como resolver os problemas de acessibilidade e b) utilizar suítes de testes apropriadas, que sejam simples e fáceis de serem usadas, por exemplo, incluindo testes automatizados.

Este capítulo visa discorrer sobre esses desafios e oferecer uma alternativa para capacitar profissionais de desenvolvimento, *design* e teste para enfrentar os desafios ainda existentes para implementar a acessibilidade em soluções *mobile*. Com foco em aplicativos Android nativos, serão explorados aqui abordagens e ferramentas para testar e melhorar a acessibilidade.

Conceitos teóricos sobre acessibilidade, com ênfase na acessibilidade em aplicativos móveis, serão explorados na Seção 3.2. Ainda nessa seção, os principais problemas de acessibilidade reportados na literatura serão discutidos, bem como os guias e recomendações disponíveis para auxiliar na eliminação das barreiras de acessibilidade e as diferentes abordagens de testes de acessibilidade. As Seções 3.3 e 3.4, por sua vez, apresentarão as principais ferramentas disponíveis, para aplicativos Android nativos, para auxiliar a realização de testes manuais, testes automatizados e análise de código. Em seguida, na Seção 3.5 este documento disponibiliza uma sugestão de atividade prática, em formato de tutorial, que visa auxiliar na identificação de problemas de acessibilidade em um aplicativo de exemplo – um aplicativo Contador — usando cada uma das ferramentas apresentadas previamente também neste documento.

1.2. Acessibilidade

Acessibilidade é o termo geral que qualifica um produto de modo a permitir que quaisquer pessoas tenham seu acesso, visando usufruir os benefícios da vida em sociedade. No contexto de desenvolvimento de software, a acessibilidade é um requisito não-funcional e uma sub característica de usabilidade [8].

A Norma ISO 9241-11 (2018) define acessibilidade como "o grau em que produtos, sistemas, serviços, ambientes e instalações podem ser usados por pessoas de uma população com a mais ampla gama de necessidades, características e capacidades de

usuários para atingir objetivos identificados em contextos de uso identificados".

No contexto da Web, a acessibilidade visa que todas as pessoas, especialmente as pessoas com deficiência e idosos, possam acessar os conteúdos dos sites em uma variedade de contextos de uso, incluindo os mais usuais, e aqueles que contam com apoio de Tecnologia Assistiva (T.A.) [9]. Considerar que diferentes usuários possuem diferentes habilidades e podem utilizar diferentes tecnologias é imprescindível para que os conteúdos sejam de fato acessíveis.

1.2.1. Diretrizes de acessibilidade

Para criar aplicativos acessíveis a todos os usuários, é imperativo seguir os padrões internacionais de acessibilidade [10]. Para contribuir com uma Web acessível a um número cada vez maior de pessoas, a Iniciativa para a Acessibilidade na Web (do Inglês *Web Accessibility Initiative* (WAI)) foi lançada em 1997 pelo W3C [11]. Como resultado desta iniciativa, foi publicada em 1999 a primeira versão das Diretrizes para Acessibilidade do Conteúdo da Web - *do Inglês: Web Content Accessibility Guidelines* (WCAG), que definem um conjunto de recomendações sobre como tonar o conteúdo da Web acessível (W3C, 1999). Na versão 2.1 da WCAG [12][1], de 2018, as diretrizes buscaram melhorar as orientações de acessibilidade para grupos específicos de usuários, incluindo usuários com deficiências que utilizam dispositivos móveis.

Um exemplo da importância dessas diretrizes na acessibilidade ao conteúdo Web é a primeira delas, que discorre sobre textos alternativos para conteúdo não textual. Por exemplo, um usuário com deficiência visual pode não conseguir visualizar uma foto apresentada em tela, mas um leitor de telas poderá ler o texto alternativo que descreve essa foto, quando fornecido, sintetizando de modo sonoro o que é lido naquele texto alternativo, para o usuário.

Para corresponder às necessidades da variedade de interessados na WCAG, o documento é estruturado em camadas de orientação que incluem [12]:

- **Princípios**. Estabelecem a base necessária para acesso ao conteúdo da Web por qualquer pessoa. São quatro os princípios:
  1. **Perceptível**. Os usuários devem ser capazes de perceber a informação apresentada;
  2. **Operável**. Os usuários devem ser capazes de operar a interface;
  3. **Compreensível**. Os usuários devem ser capazes de entender a informação apresentada e a interação esperada;
  4. **Robusto**. Os usuários devem ser capazes de acessar o conteúdo de acordo com as tecnologias atuais e futuras disponíveis, incluindo as tecnologias de apoio.
- **Diretrizes**. Procuram auxiliar para que o conteúdo seja acessível ao maior número de pessoas e adaptável às diferentes habilidades físicas, sensoriais ou cognitivas. São 13 diretrizes, associadas aos 4 princípios estabelecidos;

[1] A versão 2.2 da WCAG se tornou estável e passou a ser a versão recomendada a partir de 5 outubro de 2023. Já existe uma versão 3.0 em andamento, disponível em `https://www.w3.org/TR/wcag-3.0/`, que versa sobre interfaces IoT, para vestíveis, dispositivos móveis, entre outros, mas ainda em fase de consolidação.

- **Critério de sucesso**. Sob cada diretriz, estão alocados os critérios de sucesso, os quais são afirmações testáveis para determinar se o conteúdo satisfaz o critério. No total, são 78 critérios de sucesso que se organizam nas 13 diretrizes, e se assemelham a requisitos. Os critérios são definidos em três níveis de conformidade: A (o mais básico), AA e AAA (o mais elevado);
- **Técnicas suficientes e sugeridas**. São técnicas suficientes aquelas que, uma vez implementadas, desde que exista suporte à acessibilidade para o usuário (por exemplo, T. A. disponível), atenderá ao critério de sucesso. Um exemplo de técnicas sugeridas são as que podem melhorar a acessibilidade ao conteúdo, mas podem não ser consideradas suficientes.

1.2.2. Acessibilidade em aplicativos móveis

As diretrizes da WCAG também são aplicáveis para aplicativos móveis. Em 2015 a W3C chegou a publicar um documento com dicas sobre como aplicar o conteúdo das diretrizes no contexto *mobile*[2]. Porém, a documentação é considerada extensa para desenvolvedores, e nem sempre é simples de adaptar as diretrizes a aplicativos nativos.

Além disso, existem outros conjuntos de guias e recomendações para desenvolvedores com foco em aplicativos nativos. Por exemplo, um trabalho desenvolvido por pesquisadores do Pernambuco e disponibilizado pelo instituto Samsung Instituto de Desenvolvimento para a Informática (SIDI) reúne 48 requisitos de acessibilidade com foco em deficiência visual [13]. O documento também apresenta recomendações para designers e boas práticas para desenvolvedores, incluindo exemplos de código e uma sucinta explanação sobre como planejar testes de acessibilidade.

Outro guia disponível é o *Mobile Accessibility Standards and Guidelines*, da BBC [14]. O guia reúne melhores práticas, recomendações e padrões para aplicativos móveis e apresentam, quando aplicável, instruções para a realização de testes de acessibilidade com ferramentas de Tecnologia Assistiva.

1.2.3. Problemas de acessibilidade para pessoas com deficiência visual

Pessoas com deficiência visual usam principalmente leitores de tela para interagir com aplicativos móveis. No Android, o leitor de tela integrado à plataforma é o *Talkback*.

A usabilidade dos aplicativos para esses usuários depende diretamente da compatibilidade com os recursos de T.A. Por exemplo, o leitor de tela precisa que os elementos de UI (Interface de Usuário) sejam devidamente rotulados e organizados de forma lógica para que possam ser corretamente interpretados e a informação ser transmitida ao usuário.

Estudos recentes analisaram aplicativos Android disponíveis na *Play Store*, e identificaram que a maioria deles ainda apresenta problemas de acessibilidade [2, 3, 15]. Os problemas mais recorrentes, especialmente os que afetam usuários com deficiência visual, são de detecção e correção relativamente simples com os recursos atuais da literatura e indústria.

Além disso, um estudo de 2019 [2] identificou que 5 widgets representam 92% dos elementos de UI e respondem por 89% das violações de acessibilidade encontradas.

[2]Disponível em: `https://www.w3.org/TR/mobile-accessibility-mapping/`

A Tabela 3.1 relaciona seis desses problemas de acessibilidade mais recorrentes nos aplicativos Android avaliados nesses estudos, associados às respectivas recomendações de acessibilidade da literatura (nomeadas de maR). Além disso, pensando em soluções mais pragmáticas, foram endereçados para cada recomendação os *widgets* Android, dentre os mais utilizados, e os respectivos atributos que devem ser verificados [16]. Por exemplo, para o problema de contraste de texto insuficiente, recomenda-se utilizar uma taxa de contraste de pelo menos 4.5:1, e esse requisito deve ser aplicado a *TextViews* e *Buttons*, por exemplo.

Tabela 1.1. Problema de acessibilidade e recomendações para os *widgets* mais usados [16].

Problema	Recomendação / Requisito	Widgets
Contraste de texto insuficiente	Deve-se utilizar uma taxa de contraste de pelo menos 4.5:1 (maR7)	`TextView`, `Button`
Tamanho do alvo de toque inadequado	Alvo de toque deve ter pelo menos 48x48dp (maR17)	`Button`, `ImageButton`
Falta de rótulo no componente	Para cada controle de formulário, deve existir um *TextView* com o atributo *labelFor* associado, ou deve ter o atributo *hint* fornecido (maR10)	`EditText`, `CheckBox`, `RadioButton`, `Switch`
Imagens e/ou ícones sem alternativa textual	Todo conteúdo não textual, como imagens e conteúdo de mídia, deve possuir descrição alternativa em texto (maR1)	`ImageButton`, `ImageView`
Interação e/ou navegação confusa	Aplicativo deve suportar navegação baseada em foco (maR13)	`AccessibilityNodeInfo`
Falta de espaçamento suficiente entre os elementos	Componentes de interação devem ter um espaçamento mínimo de 8dp entre si e das bordas da tela (maR18)	

Para a realização das tarefas sugeridas na Seção 3.5, é importante compreender melhor três desses problemas.

Contraste de texto insuficiente

Usuários com baixa visão têm mais dificuldade para ler informações em uma tela quando não há contraste suficiente entre as cores do primeiro e segundo planos (fundo e fonte, por exemplo). Proporções baixas de contraste podem deixar a tela embaçada para alguns usuários, enquanto proporções mais altas deixam a tela mais nítida. Situações diferentes de luz podem aumentar as dificuldades criadas por proporções baixas de contraste. O uso de um bom contraste ajuda todos os usuários, e não apenas aqueles com deficiência.

Falta de rótulo no componente

Usuários cegos ou com baixa visão usam leitores de tela, como o *TalkBack*, para interagir com os dispositivos. O *TalkBack* anuncia o conteúdo da tela para os usuários, e esses podem interagir com ele. Quando um elemento não tem um texto associado (um *ImageButton*, por exemplo), o *TalkBack* não sabe como informar o propósito dele ao usuário. Em casos assim, o recurso pode anunciar títulos genéricos, como "Botão sem rótulo", o que não é útil para o usuário. Quando o desenvolvedor fornece um rótulo descritivo adequado, o *TalkBack* pode anunciá-lo ao usuário.

Tamanho do alvo de toque inadequado

Muitas pessoas têm dificuldade para focar em áreas de toque pequenas na tela. Isso pode acontecer por terem dedos grandes ou alguma condição médica que afeta as habilidades motoras delas. As áreas de toque pequenas também dificultam para os usuários de leitor de tela navegarem em aplicativos movendo um dedo na tela, como durante o uso do recurso "Explorar por toque" no *TalkBack*.

Em algumas circunstâncias, áreas de toque inadequadas tornam aplicativos menos acessíveis para todos os usuários, e não apenas aqueles com deficiência.

1.3. Testes automatizados no Android

A automação de testes traz benefícios como a redução do tempo de execução, uma menor propensão a erros, a liberação de recursos, a segurança para testes de regressão e o *feedback* rápido e frequente. Além disso, os testes automatizados oferecem benefícios adicionais, como impulsionar a codificação, servir como documentação e ter um bom retorno sobre o investimento [17]. Os testes automatizados aprimoram a eficiência, a qualidade e cobertura dos testes, permitindo que os desenvolvedores se concentrem em outras atividades importantes para as aplicações [18].

Os testes podem ser classificados de acordo com o tamanho ou grau de isolamento. Os testes de unidade ou testes pequenos verificam pequenas partes do código, como uma classe ou método. Para testes que precisam de mais de uma unidade de código ou usam recursos fora da unidade de código, é recomendada a realização de testes de integração ou testes médios. Já os testes de interface do usuário ou testes grandes são usados para testar fluxos de interação do usuário ou o funcionamento de partes maiores do aplicativo que precisam ser testadas [19].

Uma abordagem recomendada para testes é ilustrada na Figura 3.1. Os testes pequenos são mais baratos e fornecem um *feedback* mais rápido aos desenvolvedores, sendo preferíveis. Já os testes grandes são mais caros e demorados, mas mais confiáveis para testar o fluxo de interação do usuário [17, 19, 20].

A plataforma Android também classifica os testes de acordo com o ambiente de execução. Os testes locais são aqueles que podem ser executados diretamente no ambiente de desenvolvimento. Os testes instrumentados são testes que requerem um dispositivo Android, seja físico ou emulado, e são necessários para os testes de interface do usuário [21]. Testes de unidade instrumentados são possíveis, porém, são significativamente mais lentos do que os testes de unidade locais [20, 21].

1.4. Testes de acessibilidade no Android

Quatro abordagens são sugeridas para testes de acessibilidade, e idealmente elas devem ser combinadas. São elas:

- **Testes manuais**, que exploram o aplicativo da forma como o usuário o utilizaria. Por exemplo, usando um leitor de tela para simular o uso por usuário com deficiência visual;
- **Testes com ferramentas de análise**, que fazem verificações estáticas no código;
- **Testes automatizados**, geralmente realizados com auxílio de *frameworks* de testes;

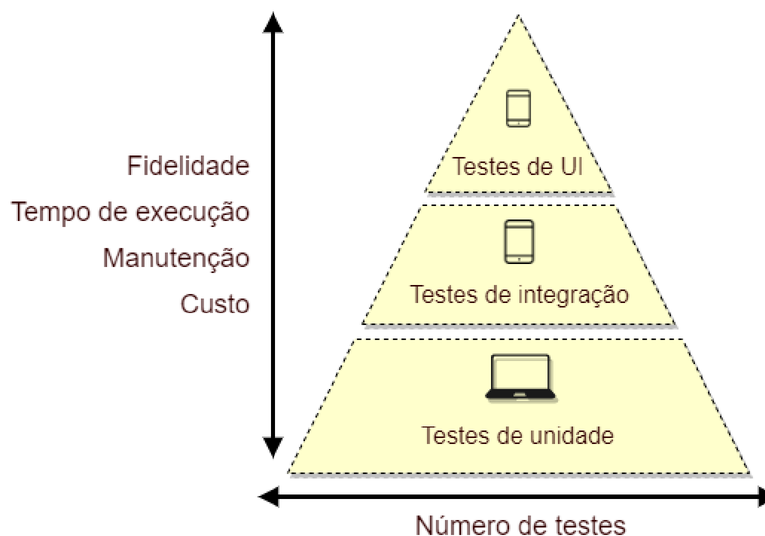

Figura 1.1. Pirâmide de sugestão para distribuição de testes de acordo com os tipos. Fonte: Adaptada da Google [21].

- **Testes de usuários**, os mais completos e ricos, pois envolvem o *feedback* de usuários reais.

Para auxiliar a criação de testes de acessibilidade, é disponibilizado no Android a biblioteca *Accessibility Test Framework*[3], uma base para outras ferramentas da plataforma, no que se refere à acessibilidade. Ela contém uma série de verificações pré-definidas, que incluem presença de *labels*, tamanho da área de toque, contraste de cores, além de outras propriedades relacionadas com serviços de acessibilidade (aqueles usados por T.A.). Além disso, também faz parte da biblioteca a API *AccessibilityChecks*[4], que permite, com poucas linhas de código, que essas verificações sejam feitas automaticamente ao realizar testes que interagem com a tela.

A Tabela 3.2 resume as principais ferramentas sugeridas na documentação para desenvolvedor Android [19] para verificações de acessibilidade. O Scanner de Acessibilidade, um aplicativo que analisa a tela de outros aplicativos, é o único que foi criado especificamente para a acessibilidade. *UI Automator* [22] e *Espresso* [23] são ferramentas para testes automatizados de UI, que se diferem pelo tipo de teste: o *UI Automator* realiza testes *blackbox*, já o Espresso realiza testes *whitebox*. Como o *Espresso* é possível acionar a API *AccessibilitChecks*. Todos esses recursos requerem um dispositivo para serem executados.

Uma opção para testes locais é o *Roboletric* [24]. Ele chegou a suportar testes com a *AccessibilityChecks*, porém essa funcionalidade foi descontinuada na versão 4.5, pois não era capaz de identificar problemas que eram identificados com o *Espresso*. O *Roboletric* suporta alguma interação com elementos da UI, por simulações, porém o suporte a gráficos nativos só foi adicionado na versão 4.10, de abril de 2023.

[3]Disponível em: `https://github.com/google/Accessibility-Test-Framework-for-Android`

[4]Disponível em: `https://developer.android.com/training/testing/espresso/accessibility-checking?hl=pt-br`

Tabela 1.2. Ferramentas para testar acessibilidade no Android.

Ferramenta	Testes	Execução	Esforço requerido
Scanner de Acessibilidade	Teste grandes	Instrumentado	Execução e verificação do *app* tela a tela
UI Automator	Testes grandes	Instrumentado	Automatização de testes de UI
Espresso	Testes médios Testes grandes	Instrumentado	Chamada da API *AccessibilityChecks* a partir de *View* raíz de cada tela
Roboletric	Testes pequenos	Local	Escrita de testes para cada verificação de acessibilidade
Lint	Estáticos	Local	Inspeção automática em conjunto pré-definido de verificações

Por último, o Android *Lint* [25] é uma ferramenta de análise estática de código integrada à IDE de desenvolvimento.

Além das ferramentas da própria plataforma Android, podem ser encontradas na literatura uma série de outras ferramentas que podem ser utilizadas por desenvolvedores para testar a acessibilidade de aplicativos Android, como **PUMA** [26], **forApp**[5], **Axe Android**[6], ***IBM Mobile Accessibility Checker***[7], **Latte** [27], **Bility** [28], **Mate** [29] e **AccessibiLint** [30].

Para as tarefas sugeridas neste capítulo e descritas na Seção 3.5, a seguir, são utilizados o **Scanner de Acessibilidade**, o **Espresso** e o ***Automated Accessibility Tests Kit (AATK)*** [16]. Esse último, um kit de testes locais de acessibilidade para aplicativos Android. Trata-se de uma biblioteca escrita em Java, com testes escritos para serem executados com o *Roboletric*, ou seja, não requerem uso de dispositivos.

1.5. Sugestão de atividade prática para testes

Nesta seção são apresentadas três alternativas de ferramentas disponíveis para auxiliar a realização de testes de acessibilidade em aplicativos Android nativos. O leitor é guiado a identificar problemas de acessibilidade em um aplicativo de exemplo – um app Contador – com cada uma das seguintes ferramentas:

- **Scanner de Acessibilidade (*Accessibility Scanner*)**: ferramenta do Google para verificar aplicativos móveis em busca de problemas de acessibilidade;
- **Espresso**: ferramenta de automação de testes de interface do usuário no Android, para testes instrumentados;
- **AATK**: biblioteca Android contendo testes de acessibilidade automatizados locais, ou seja, que não requerem uso de dispositivo físico ou emulado [16].

Para cada ferramenta são fornecidas as instruções passo-a-passo para preparação, configuração, escrita dos testes (quando aplicável), execução e visualização dos resultados. Ao final de cada uma delas, também são oferecidas sugestões para solucionar os problemas encontrados e re-executar os testes.

Os recursos necessários para a execução das tarefas são:

[5] `https://www.forapp.org/`
[6] `https://www.deque.com/android-accessibility/`
[7] `https://www-03.ibm.com/able/mobile-accessibility-checker.html`

- Estação de trabalho com Android Studio[8] instalado (versão 2021.1.1 ou superior);
- Dispositivo:
  - smartphone Android com cabo USB e modo de desenvolvedor habilitado; ou
  - emulador com Play Store baixado e configurado no Android Studio (sugestão: Pixel 2 API 30).

Não é necessário nenhum conhecimento prévio sobre acessibilidade ou testes automatizados para realizar essas tarefas. No entanto, se assume que o interessado seja capaz de baixar um projeto do GitHub e abrir no Android Studio.

A realização deste treinamento permite:

- Compreender os conceitos básicos de acessibilidade em sistemas computacionais;
- Aprender a utilizar ferramentas para testar a acessibilidade em aplicativos Android nativos;
- Identificar, solucionar e prevenir problemas de acessibilidade em um aplicativo de exemplo por meio de testes de acessibilidade automatizados.

1.5.1. O aplicativo Contador

Neste treinamento é utilizado um aplicativo existente, o Contador, derivado do Google Codelabs.[9] Esse aplicativo permite aos usuários rastrear, incrementar e decrementar uma contagem numérica. Embora o aplicativo seja simples, existem nele alguns problemas de acessibilidade que podem dificultar que usuários com deficiência interajam com sua interface.

O código-fonte da versão inicial do aplicativo pode ser obtido no link `https://github.com/AALT-Framework/poor-accessibility-apps`.

Para executar testes de acessibilidade e identificar os problemas com cada uma das três ferramentas, para simplificação deste tutorial, devem ser criadas três cópias da pasta do projeto localmente.

1. Abra o terminal ou prompt de comando no diretório onde deseja clonar o projeto;
2. Clone o repositório disponível em `https://github.com/AALT-Framework/poor-accessibility-apps/`;
3. Crie uma cópia da pasta "Contador", que está dentro do repositório clonado, para cada projeto. Nomeie cada cópia de acordo com a ferramenta que se deseja testar, da seguinte forma:
   - Contador-AccessibilityScanner
   - Contador-Espresso
   - Contador-AATK
4. Para cada ferramenta, abra a respectiva pasta do projeto no Android Studio e siga as instruções do tutorial.

[8] `https://developer.android.com/studio`
[9] `https://codelabs.developers.google.com/`

1.5.2. Testes de acessibilidade no aplicativo Contador com o Scanner de Acessibilidade

O Scanner de acessibilidade (*Accessibility Scanner*) é uma ferramenta criada pelo Google para sugerir melhorias de acessibilidade em aplicativos Android, como aumento de áreas de toque pequenas, aumento de contraste e descrições de conteúdo. As melhorias possibilitam que pessoas com deficiência possam usar os aplicativos com mais facilidade.

Preparação

1. No Android Studio, abra o projeto do aplicativo Contador, da pasta *Contador-AccessibilityScanner*, criada no 2º passo deste treinamento;
2. Execute o projeto no dispositivo Android ou emulador. O aplicativo Contador será aberto;
3. Siga estas etapas para fazer o *download* e configurar o Scanner de Acessibilidade:
   (a) Abra o aplicativo Play Store no dispositivo Android;
   (b) Faça o download do Scanner de Acessibilidade na Google Play Store;
   (c) Após instalar o Scanner de Acessibilidade, navegue até Configurações ¿ Acessibilidade no dispositivo. Localize e ative o Scanner de acessibilidade (toque em ”Permitir ou ”OK”e, em seguida, em ”Iniciar autorização”para concluir o fluxo de configuração);
   (d) Retorne ao aplicativo Contador. A tela deverá estar como a ilustrada na Figura 3.2.

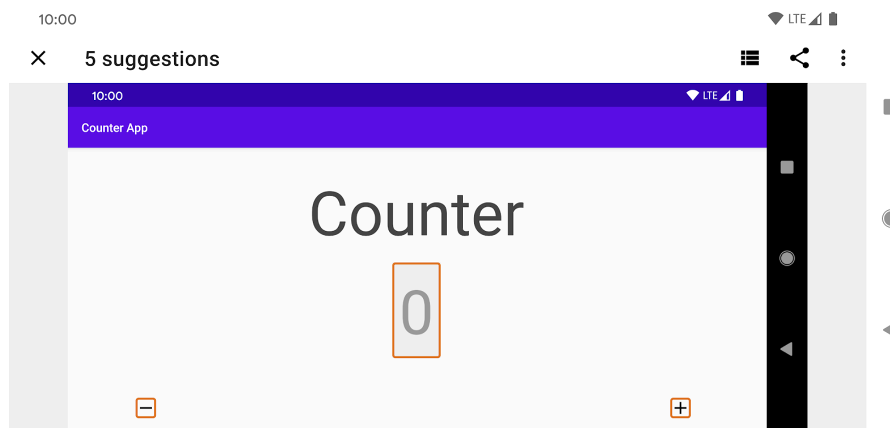

Figura 1.2. Tela do aplicativo Contador após habilitar Scanner de Acessibilidade.

O Scanner de acessibilidade cria um botão de ação flutuante azul (FAB, na sigla em Inglês), que fica sobreposto ao conteúdo na tela.

1. Toque no FAB para iniciar a verificação. Ao fazer isso, o Scanner de acessibilidade examina a interface de usuário da tela, realiza uma auditoria rápida de acessibilidade e prepara as sugestões de melhoria;
2. Toque em ”Verificar”para ver as sugestões de melhoria. O Scanner de acessibilidade exibe uma lista de sugestões de melhoria de acessibilidade, classificadas por prioridade. As sugestões de melhoria são baseadas nas diretrizes de acessibilidade do Google;

3. Toque em uma sugestão de melhoria para ver mais detalhes. Por exemplo, toque em ”Aumentar o contraste”para ver mais detalhes sobre essa sugestão.

O Scanner de acessibilidade tem cinco sugestões para melhorar a acessibilidade do Contador. Nos passos a seguir são sugeridas alternativas para realizar essas melhorias.

1.5.3. Como garantir um contraste de cor adequado

No Contador, o contraste de cor é simples de melhorar. A *TextView*, que mostra a contagem, usa um plano de fundo cinza-claro com texto cinza:

```
<TextView
    ...
    android:background="@color/lightGrey"
    android:textColor="@color/grey"
    ...
 />
```

O desenvolvedor pode remover o plano de fundo, escolher outro mais claro ou deixar o texto mais escuro. Nesta atividade, é sugerido escolher uma cor mais escura para o texto. Segue abaixo exemplos de cores que foram definidas em **colors.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    ...
    <color name="lightGrey">#EEEEEE</color>
    <color name="grey">#999999</color>
    <color name="darkGrey">#666666</color>
</resources>
```

Abra o arquivo **res/layout/activity_main.xml** e mude *android:textColor=”@color/grey”* para *android:textColor=”@color/darkGrey”*:

```
<TextView
    ...
    android:background="@color/lightGrey"
    android:textColor="@color/darkGrey"
    ...
 />
```

Agora, execute o aplicativo e veja o contraste melhorado. A proporção de contraste agora é de 4.94:1, consideravelmente melhor que 2.45:1, que é o valor da taxa de contraste anterior.

Diretrizes de acessibilidade de conteúdo da Web [12] recomendam uma proporção de contraste mínima de 4.5:1 para todo o texto. A proporção de 3.0:1 é considerada aceitável para textos grandes ou em negrito.

Pressione o FAB para iniciar outra verificação no Scanner de acessibilidade. É possível ver que o aplicativo não tem mais sugestões relacionadas ao contraste de cor.

1.5.4. Como adicionar rótulos ausentes

No aplicativo Contador, as ações de decrementar e incrementar são representadas por dois *ImageButtons*, (-) e (+) respectivamente. Por serem imagens sem rótulos, um leitor de tela como o *TalkBack* não consegue comunicar adequadamente a semântica das visualizações para o usuário, anunciando simplesmente "botão sem rótulo", quando um desses botões é focado.

Para corrigir esse problema, atribua uma *android:contentDescription* para cada botão:

```
<ImageButton
    android:id="@+id/subtract_button"
    ...
    android:contentDescription="@string/decrement" />

<ImageButton
    android:id="@+id/add_button"
    ...
    android:contentDescription="@string/increment" />
```

Use *strings* localizadas nas descrições de conteúdo. Assim, elas poderão ser adequadamente traduzidas. Para esta atividade, as *strings* já foram definidas em **res/values/strings.xml**.

Agora, um leitor de tela pode anunciar o valor da *contentDescription* fornecida (adequadamente traduzida para o idioma local) quando o usuário foca nos botões.

Execute o Scanner de acessibilidade novamente. Não deverá mais ter sugestões relacionadas a rótulos ausentes.

1.5.5. Como aumentar áreas de toque

O Scanner de acessibilidade continua sugerindo que os botões (-) e (+) precisam ter uma área de toque maior. a sugestão de ajuste é a seguinte.

Os dois botões no Contador são pequenos (24dp x 24dp). No geral, o tamanho adequado para a área de toque de itens focáveis precisa ser, pelo menos, de 48dp x 48dp. Se for possível criar uma área ainda maior, melhor. Ao aumentar a área de toque de 24dp x 24dp para 48dp x 48dp, ela é expandida por um fator de 4.

O desenvolvedor/designer tem várias opções para aumentar a área de toque dos botões. Por exemplo, escolher uma das opções abaixo:

- Adicionar *padding* em volta dos ícones;
- Adicionar um *minWidth* e/ou *minHeight* (os ícones ficarão maiores);
- Registrar um *TouchDelegate*.

Adicione um pouco de *padding* a cada visualização:

```
<ImageButton
    ...
    android:padding="@dimen/icon_padding"
```

```
    ... />

<ImageButton
    ...
    android:padding="@dimen/icon_padding"
   ... />
```

O valor de *@dimen/icon_padding* está definido como 12dp (veja res/dimens.xml). Quando o padding é aplicado, a área de toque do controle se torna 48dp x 48dp (24dp + 12dp em cada direção).

Execute o aplicativo novamente para confirmar os novos limites de layout. Agora, a área de toque dos botões é maior.

Execute o Scanner de acessibilidade novamente. Desta vez, a análise será concluída sem sugestões.

Como desativar o Scanner de Acessibilidade

Após concluídos os testes com o Scanner de Acessibilidade, caso se deseje desativá-lo, navegue para **Configurações**, **Acessibilidade** e defina o Scanner de acessibilidade como **Desativado**.

1.5.6. Testes de Acessibilidade no Aplicativo Contador com o Espresso

O Espresso é um *framework* de teste de interface do usuário para aplicativos Android, permitindo que os desenvolvedores criem testes automatizados para interagir com os elementos da interface do usuário do aplicativo. É integrado com o Android Studio e pode ser executado em dispositivo físico ou emulado.

Para testar acessibilidade com o Espresso, é possível usar a API *AccessibilityChecks* do ATF.

Preparação

No Android Studio, abra o projeto do aplicativo Contador, da pasta Contador-Espresso criada no 2º passo deste treinamento.

O desenvolvedor precisará de uma nova dependência para o pacote *androidTestImplementation*. Confira se a linha seguinte já foi adicionada no arquivo **app/build.gradle**.

1. Edite o arquivo *build.gradle*[10] da raiz, adicionando a seguinte linha na lista de dependências;
2. Depois de fazer essas alterações, sincronize seu projeto para garantir que elas entrem em vigor.

[10]**Onde encontro isso?** Ao abrir o projeto no Android Studio, na visualização Android (painel esquerdo), há uma seção *Gradle Scripts*. Dentro, há um arquivo chamado **build.gradle (Módulo: Contador.app)**, ou algo parecido.

```
dependencies {
  ...
  androidTestImplementation
    'androidx.test.espresso:espresso-accessibility:3.3.0-alpha05'
  ...
}
```

Crie uma classe de teste instrumentado para a tela principal

1. No Android Studio, abra o painel *Project* e encontre esta pasta:
   - **com.example.contador (androidTest)**;
2. Clique com o botão direito na pasta contador e selecione *New* - *Java Class*;
3. Nomeie como *MainActivityInstrumentedTest*. Assim é possível saber que esta classe de teste instrumentado se refere à *MainActivity*.

Com a classe *MainActivityInstrumentedTest* gerada e aberta, crie seu primeiro teste. Para o propósito deste treinamento, será escrito apenas um único teste, que verifica se o código para incrementar a contagem funciona corretamente (por questões de brevidade, o teste para decrementar a contagem foi omitido). Sua classe deverá ficar assim:

```
public class MainActivityInstrumentedTest {
  @Rule
  public ActivityScenarioRule<MainActivity>
    mActivityTestRule =
      new ActivityScenarioRule<>(MainActivity.class);

  @Test
  public void testIncrement(){
    Espresso.onView(withId(R.id.add_button))
      .perform(ViewActions.click());
    Espresso.onView(withId(R.id.countTV))
      .check(matches(withText("1")));
  }
}
```

Agora, verifique se o seu computador está conectado a um dispositivo com a depuração USB ativada, e execute os testes clicando no botão de seta verde imediatamente à esquerda de *@Test public void testIncrement()*. Se um dispositivo físico conectado via USB estiver sendo usado, se certifique de que o dispositivo esteja desbloqueado e com a tela ligada. Observe que pressionar Ctrl+Shift+F10 (Control+Shift+R em um Mac) executa os testes no arquivo atualmente aberto.

O teste deve ser executado até o final e deve passar, confirmando que o incremento da contagem funciona como esperado.

A seguir, é preciso modificar o teste para verificar também a acessibilidade.

Habilite as checagens de acessibilidade com o Espresso

Com o Espresso, é possível habilitar verificações de acessibilidade chamando *AccessibilityChecks.enable()* de um método de configuração. Adicionar essa única linha de código permite que seja feito o teste da interface do usuário para acessibilidade, tornando fácil integrar a verificação de acessibilidade em seu conjunto de testes.

Para configurar a classe **MainActivityInstrumentedTest** para checagens de acessibilidade, adicione o seguinte método de configuração antes do teste.

```
@BeforeClass
public static void beforeClass()
{
    AccessibilityChecks.enable();
}
```

Agora, execute o teste novamente. Desta vez, será possível perceber que o teste falha. No painel *Run*, clique duas vezes em *testIncrement* para ver os resultados. Será possível notar a mensagem de erro apontando dois problemas de acessibilidade:

- O *ImageButton* de adição (+) contém uma imagem, mas não tem um rótulo;
- O *ImageButton* de adição (+) precisa de um alvo de toque maior.

Essas verificações de acessibilidade estão associadas ao botão "adicionar", que é a visualização na qual foi executada a ação do teste de incremento. Para permitir que a API examine outras visualizações na hierarquia, sem precisar executar ações adicionais em outras visualizações, é necessário chamar o método *setRunChecksFromRootView*, do objeto *AccessibilityValidator* na habilitação da *AccessibilityChecks*.

Modifique seu método da seguinte forma:

```
@BeforeClass
public static void beforeClass()
{
    AccessibilityChecks.enable()
        .setRunChecksFromRootView(true);
}
```

Execute os testes novamente. Desta vez, os problemas de acessibilidade encontrados pelo Espresso, com o *AccessibilityChecks*, no aplicativo Contador, são os mesmos identificados na Seção 3.5.2, com o Scanner de Acessibilidade, incluindo as falhas encontradas no botão de decremento (-) e no *TextView* de visualização da contagem. Portanto, para corrigi-los, devem ser realizadas as mesmas tarefas indicadas nas seções 3.5.3, 3.5.4 e 3.5.5.

Após todas as correções, execute o teste novamente. O teste deve executar com sucesso até o final.

1.5.7. Testes de acessibilidade no aplicativo Contador com o AATK

O Kit de Testes de Acessibilidade Automatizados (do inglês *Automated Accessibility Tests Kit* (AATK)) para Aplicativos Android consiste em uma coleção de testes de acessibili-

dade automatizados projetados para serem executados com o *Robolectric*. Isso permite que sejam executados como testes locais, sem a necessidade de um dispositivo físico ou emulado.

Este kit foi desenvolvido com foco nos problemas de acessibilidade mais comuns, para pessoas com deficiência visual, e nos *widgets* mais usados, em que muitos problemas de acessibilidade tendem a ocorrer.

Preparação

No Android Studio, abra o projeto do aplicativo Contador, da pasta Contador-AATK, criada no 2º passo deste treinamento.

Siga os seguintes passos para preparar o projeto para adicionar testes de acessibilidade automatizados:

1. Edite o arquivo **settings.gradle** da raíz[11], adicionando *maven url 'https://jitpack.io'* na lista de *repositories*:

   ```
   dependencyResolutionManagement {
       repositoriesMode
           .set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
       repositories {
           google()
           mavenCentral()
           maven { url 'https://jitpack.io' }
       }
   }
   ```

2. Configure seu arquivo **build.gradle** no nível do aplicativo[12] para habilitar a execução de testes com o *Robolectric* e o AATK, atualizando os *testOptions* e adicionando as dependências necessárias.
   Primeiro, adicione a diretiva *testOptions* com as seguintes linhas, dentro da diretiva *android*, assim:

   ```
   android{
     ...
     testOptions {
       // Usado para testar elementos dependentes do Android
       // na pasta de teste
       unitTests.includeAndroidResources  = true
       unitTests.returnDefaultValues = true
     }
   }
   ```

   Em seguida, adicione estas duas dependências como *testImplementation*:

   ```
   android{
     ...
   ```

[11] **Onde encontro isso?** Ao abrir o projeto no Android Studio, na visualização do Android (painel esquerdo), há uma seção *Gradle Scripts*. Dentro, há um arquivo chamado **settings.gradle (Project: Settings)**.

[12] **Onde encontro isso?** Ao abrir o projeto no Android Studio, na visualização Android (painel esquerdo), há uma seção *Gradle Scripts*. Dentro, há um arquivo chamado **build.gradle (Módulo: Contador.app)**, ou algo parecido.

```
        testOptions {
          dependencies {
            ...
            testImplementation 'org.robolectric:robolectric:4.9'
            testImplementation
              'com.github.AALT-Framework:
              android-accessibility-test-kit:v1.0.0-alpha'
            ...
          }
        }
      }
```

3. Depois de fazer essas alterações, sincronize seu projeto para garantir que elas entrem em vigor.

Crie uma classe de teste local para a tela principal

1. No Android Studio, abra o painel *Project* e encontre esta pasta:
   - **com.example.contador (test)**
2. Clique com o botão direito na pasta contador e selecione *New* - *Java Class*;
3. Nomeie como *MainActivityTest*. Assim será possível saber que essa classe de teste se refere à *MainActivity*.

Com a classe **MainActivityTest** gerada e aberta, comece a configurá-la para executar os testes AATK.

Sua classe deverá ficar assim:

```
@RunWith(RobolectricTestRunner.class)
public class MainActivityTest {
  private View rootView;
  private AccessibilityTestRunner runner;

  @Rule
  public ErrorCollector collector = new ErrorCollector();

  @Before
  public void setUp() {
    // Crie uma instancia da atividade
    MainActivity activity =
      Robolectric
        .buildActivity(MainActivity.class)
        .create()
        .get();

    // Obtenha a view raiz da hierarquia de exibicao
    rootView = activity
      .getWindow()
      .getDecorView()
      .getRootView();
    runner = new AccessibilityTestRunner(collector);
  }
```

```
}
```

O que foi feito:

1. Adicionado o escopo da classe para executar com *RoboletricTestRunner*;
2. Declarado um atributo privado para manter o *rootView* e o *AccessibilityTestRunner*;
3. Declarada uma propriedade pública para o *ErrorCollector*;
4. Adicionado um método *setUp* que habilita a execução do kit em qualquer novo teste criado.

Escreva seu primeiro teste com o AATK

Adicione um método de teste para cada teste de acessibilidade que deseja executar. Será iniciado com a verificação da taxa de contraste de cores.

O desenvolvedor pode utilizar o teste de taxa de contraste do AATK (*TestAdequateContrastRatio*) da seguinte forma:

1. Adicione um método de teste. Procure seguir boas convenções para nomenclatura de testes. Por exemplo: **deve_UsarTaxaDeContrasteAdequada**;
2. Chame o método *runAccessibilityTest* do executor do kit, passando como parâmetro a *view* raíz e uma nova instância do teste desejado:

```
@Test
public void deve_UsarTaxaDeContrasteAdequada(){
  runner.runAccessibilityTest(rootView,
    new TestAdequateContrastRatio());
}
```

3. Execute seu teste. Clique com o botão direito do mouse sobre o nome do método e selecione *Run MainActivityTest.deve_UsarTaxaDeContrasteAdequada*;
4. No painel *Run*, clique duas vezes em *deve_UsarTaxaDeContrasteAdequada* para ver os resultados. Será possível notar a mensagem de erro, a identificação da visualização, a taxa esperada e a taxa atual;
5. Abra o arquivo **res/layout/activity_main.xml**, encontre o *TextView* e altere *android:textColor="@color/grey"* para *android:textColor="@color/darkGrey"*;
6. Volte ao item 3 para refazer o teste e ver se ele passou.

1.5.7.1. Escreva novos testes

Agora que já criou seu primeiro teste, é possível adicionar outros. A seguir, é sugerido mais dois exemplos. Consulte a documentação do AATK, disponível em `https://github.com/AALT-Framework/android-accessibility-test-kit`, para ter acesso a todos os testes disponíveis.

Para testar conteúdos não textuais sem descrição alternativa, será preciso utilizar o teste de texto alternativo do AATK (*TestMustHaveAlternativeText*), assim como foi feito para o teste de contraste.

1. Adicione um método de teste. Por exemplo: *deve_ConterAlternativaTextual*;

2. Chame o método *runAccessibilityTest* do executor do kit, passando como parâmetro a *view* raíz e uma nova instância do teste desejado:

```
@Test
public void deve_ConterAlternativaTextual(){
  runner.runAccessibilityTest(rootView,
    new TestMustHaveAlternativeText());
}
```

3. Execute seu teste. Verifique os resultados. Realize as correções, conforme visto na Seção 3.5.4. Reexecute o teste.

Para verificar o tamanho dos alvos de toque, será preciso utilizar o teste de texto alternativo do AATK (*TestTouchTargetSize*), assim como foi feito para os testes anteriores.

1. Adicione um método de teste. Por exemplo: *deve_AlvoDeToquePossuirTamanhoMinimo*;
2. Chame o método *runAccessibilityTest* do executor do kit, passando como parâmetro a *view* raíz e uma nova instância do teste desejado:

```
@Test
public void deve_AlvoDeToquePossuirTamanhoMinimo(){
  runner.runAccessibilityTest(rootView,
    new TestTouchTargetSize());
}
```

3. Execute seu teste. Verifique os resultados. Realize as correções, conforme visto na Seção 3.5.5. Re-execute o teste.

1.6. Considerações Finais

A acessibilidade em soluções computacionais, além de ser um imperativo ético, é uma condição legal no contexto brasileiro. A empatia com essa temática deve ser complementada por uma abordagem profissional e legalmente respaldada, exigindo que os profissionais envolvidos no desenvolvimento de tecnologias compreendam e implementem de forma séria os requisitos de acessibilidade, desde a concepção do projeto e não como um requisito adicional.

Apesar dos recursos disponíveis e da crescente conscientização sobre a importância da acessibilidade, muitos aplicativos Android ainda apresentam barreiras de acessibilidade recorrentes. Essas barreiras, contudo, possuem soluções relativamente simples com os recursos atuais disponíveis, sugerindo que a implementação de práticas que conduzam à acessibilidade pode ser mais disseminada e efetiva.

É fundamental oferecer alternativas tangíveis para o desenvolvimento de aplicativos acessíveis, proporcionando aos profissionais da área ferramentas e conhecimentos que possibilitem a criação de soluções inclusivas. Neste contexto, o presente documento desempenha um papel relevante ao apresentar, de maneira prática, a utilização dos recursos existentes para solucionar os problemas mais comuns de acessibilidade em aplicativos Android. A partir do estabelecimento de prioridades, com base na literatura, são oferecidas tarefas passo-a-passo com o objetivo de capacitar os desenvolvedores na promoção de aplicativos Android nativos, com especial foco nos usuários com deficiência visual.

Este tutorial também contribui para a disseminação e fomento da acessibilidade

na comunidade de desenvolvedores. Ao oferecer uma abordagem prática e aplicável, que busca influenciar positivamente a mentalidade e as práticas dos profissionais de tecnologia.

Referências

[1] BRASIL. Lei nº 13.146, de 6 de julho de 2015 (lei brasileira de inclusão da pessoa com deficiência / estatuto da pessoa com deficiência), 7 2015. URL http://www2.camara.leg.br/legin/fed/lei/2015/lei-13146-6-julho-2015-781174-normaatualizada-pl.pdf.

[2] S. Yan and P. G. Ramachandran. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing*, 12(1), 2 2019. ISSN 19367228. doi: 10.1145/3300176. URL https://doi.org/10.1145/3300176.

[3] Alberto Dumont Alves Oliveira, Paulo Sérgio Henrique Dos Santos, Wilson Estécio Marcílio Júnior, Wajdi M Aljedaani, Danilo Medeiros Eler, and Marcelo Medeiros Eler. Analyzing accessibility reviews associated with visual disabilities or eye conditions. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3581315. URL https://doi.org/10.1145/3544548.3581315.

[4] Yavuz Inal, Kerem Rızvanoğlu, and Yeliz Yesilada. Web accessibility in Turkey: awareness, understanding and practices of user experience professionals. *Universal Access in the Information Society*, 18(2):387–398, June 2019. ISSN 1615-5297. doi: 10.1007/s10209-017-0603-3. URL https://doi.org/10.1007/s10209-017-0603-3.

[5] M. V. Rodrigues Leite, L. P. Scatalon, A. P. Freire, and M. M. Eler. Accessibility in the mobile development industry in brazil: Awareness, knowledge, adoption, motivations and barriers. *Journal of Systems and Software*, 177:110942, 7 2021. ISSN 0164-1212. doi: 10.1016/J.JSS.2021.110942.

[6] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford. Accessibility in software practice: A practitioner's perspective. *Trans. on Software Engineering and Methodology*, 31:1–26, 10 2022. ISSN 1049-331X. doi: 10.1145/3503508. URL https://dl.acm.org/doi/10.1145/3503508.

[7] Darliane Miranda and João Araujo. Studying industry practices of accessibility requirements in agile development. In *Proc of the 37th ACM/SIGAPP Symposium on Applied Computing*, volume 10, page 1309–1317. Association for Computing Machinery, 4 2022. ISBN 9781450387132. doi: 10.1145/3477314.3507041. URL https://doi.org/10.1145/3477314.3507041.

[8] ISO 9241-11. Ergonomics of human-system interaction – part 11: Usability: Definitions and concepts. Technical report, ISO - International Organization for Standardization, 2018.

[9] Helen Petrie, Andreas Savva, and Christopher Power. Towards a unified definition of web accessibility. In *Proc of the 12th {Web} for {All} {Conference}*, pages 35:1–35:13. ACM, 2015. ISBN 978-1-4503-3342-9. doi: 10.1145/2745555.2746653. URL http://doi.acm.org/10.1145/2745555.2746653.

[10] Amina Bouraoui and Imen Gharbi. Model driven engineering of accessible and multi-platform graphical user interfaces by parameterized model transformations. *Science of Computer Programming*, 172:63–101, 2019. ISSN 01676423. doi: 10.1016/j.scico.2018.11.002. URL https://linkinghub.elsevier.com/retrieve/pii/S0167642318304301.

[11] Daniel Dardailler. Wai early days, 2009. URL https://www.w3.org/WAI/history.

[12] W3C. Web content accessibility guidelines (wcag) 2.1, 2018. URL https://www.w3.org/TR/WCAG21/#identify-input-purpose.

[13] Clauirton Siebra, Tatiana B Gouveia, Jefte Macedo, Fabio Q B Da Silva, Andre L M Santos, Walter Correia, Marcelo Penha, Fabiana Florentin, and Marcelo Anjos. Toward accessibility with usability: Understanding the requirements of impaired uses in the mobile context. In *Proc of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017*, pages 6:1–6:8, New York, NY, USA, 2017. ACM. ISBN 9781450348881. doi: 10.1145/3022227.3022233. URL http://doi.acm.org/10.1145/3022227.3022233.

[14] BBC. Accessibility - bbc, 2022. URL https://www.bbc.co.uk/accessibility/.

[15] A. Alshayban, I. Ahmed, and S. Malek. Accessibility issues in android apps: State of affairs, sentiments, and ways forward. In *Proc of the 42nd Intl Conf on Software Engineering*, ICSE '20, page 1323–1334, New York, NY, USA, 2020. ACM. ISBN 9781450371216. doi: 10.1145/3377811.3380392. URL https://doi.org/10.1145/3377811.3380392.

[16] Anderson Canale Garcia, Silvana Maria Affonso de Lara, Lianna Mara Castro Duarte, Renata Pontin de Mattos Fortes, and Kamila Rios Da Hora Rodrigues. Early accessibility testing – an automated kit for android developers. In *Proceedings of the 29th Brazilian Symposium on Multimedia and the Web*, WebMedia '23, page 11–15, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400709081. doi: 10.1145/3617023.3617028. URL https://doi.org/10.1145/3617023.3617028.

[17] Lisa Crispin and Janet Gregory. *Agile testing: A practical guide for testers and agile teams*. Pearson Education, Boston, MA, 2009. ISBN 978-0-321-53446-0.

[18] Márcio Eduardo Delamaro, José Carlos Maldonado, and Mário Jino. *Introdução ao teste de software*. Elsevier, Rio de Janeiro, 2016. ISBN 978-85-352-8352-5.

[19] Google. Developer guides, 2023. URL https://developer.android.com/guide/.

[20] M Aniche. *Real World Test-Driven Development*. Casa do Código, São Paulo, 2014. ISBN 9788566250572.

[21] Google. Fundamentals of testing android apps, 2023. URL `https://developer.android.com/training/testing/fundamentals`.

[22] Google. Write automated tests with ui automator, 2023. URL `https://developer.android.com/training/testing/ui-testing/uiautomator-testing`.

[23] Google. Espresso, 2023. URL `https://developer.android.com/training/testing/espresso`.

[24] Google. Roboletric. `https://github.com/robolectric/robolectric`, 2023.

[25] Google. Improve your code with lint checks, 2023. URL `https://developer.android.com/studio/write/lint`.

[26] Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, and Ramesh Govindan. Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, page 204–217, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327930. doi: 10.1145/2594368.2594390. URL `https://doi.org/10.1145/2594368.2594390`.

[27] Navid Salehnamadi, Abdulaziz Alshayban, Jun-Wei Lin, Iftekhar Ahmed, Stacy Branham, and Sam Malek. Latte: Use-case and assistive-service driven automated accessibility testing framework for android. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445455. URL `https://doi.org/10.1145/3411764.3445455`.

[28] Aaron R. Vontell. *Bility : Automated Accessibility Testing for Mobile Applications*. PhD thesis, Massachusetts Institute of Technology, 2019. URL `https://dspace.mit.edu/handle/1721.1/121685`.

[29] M. M. Eler, J. M. Rojas, Yan Ge, and G. Fraser. Automated accessibility testing of mobile apps. In *2018 IEEE 11th Intl Conf on Software Testing, Verification and Validation (ICST)*, pages 116–126, Västerås, Sweden, 5 2018. IEEE Inc. ISBN 9781538650127. doi: 10.1109/ICST.2018.00021.

[30] Arthur Floriano Barbosa Andrade de Oliveira and Lucia Vilela Leite Filgueiras. Accessibilint: A tool for early accessibility verification for android native applications. In *Procngs of the 18th Brazilian Symposium on Human Factors in Computing Systems*. Association for Computing Machinery, 2019. ISBN 9781450369718. doi: 10.1145/3357155.3360474. URL `https://doi.org/10.1145/3357155.3360474`.