Transferindo responsabilidades: um relato de como é possível capacitar desenvolvedores para tornar os aplicativos Android mais acessíveis

Anderson C. Garcia[1], **Renata P. M. Fortes**[1], **Kamila R. H. Rodrigues**[1]

[1]Instituto de Ciências Matemáticas e de Computação ICMC / USP
Caixa Postal 15.064 – 91.501-970 – São Carlos – SP – Brazil

andersongarcia@usp.br, {renata,kamila.rios}@icmc.usp.br

Resumo. ***Introdução****: A acessibilidade de sistemas interativos visa possibilitar que pessoas com dificuldades e/ou deficiências acessem recursos tecnológicos e realizem suas tarefas da mesma forma que os demais usuários. Com a popularização dos dispositivos móveis, criar aplicações acessíveis tornou-se essencial. Existem diversos conjuntos de diretrizes e ferramentas disponíveis para testes de acessibilidade. No entanto, desenvolvedores ainda enfrentam barreiras ao considerar a acessibilidade durante o desenvolvimento de aplicativos móveis.* ***Objetivo****: Neste trabalho, investigamos o estado atual dos testes de acessibilidade em apps Android e compartilhamos nossas experiências com as ferramentas e técnicas atualmente disponíveis para criação de apps acessíveis.* ***Resultados:*** *Nossa análise revelou diversas limitações, incluindo o foco restrito a um subconjunto pequeno de problemas de acessibilidade, a falta de integração de testes locais ao ambiente de desenvolvimento e a atribuição inadequada de responsabilidades aos desenvolvedores. Defendemos a adoção de práticas já familiares a esses profissionais, como a integração dos testes de acessibilidade ao fluxo de desenvolvimento e a oferta de documentação e capacitação adequadas. Essas práticas podem potencializar os desenvolvedores na criação de apps acessíveis.*
Palavras-Chave *Acessibilidade, Aplicativos Android, Desenvolvedores, Boas práticas.*

1. Introdução

Aplicativos de dispositivos móveis podem ajudar as pessoas a realizar tarefas diárias. No entanto, pessoas com deficiências podem enfrentar barreiras ao usar os recursos desses dispositivos se eles não fornecerem acessibilidade adequada. A acessibilidade é o grau em que produtos, sistemas, serviços, ambientes e instalações podem ser utilizados por pessoas de uma população, com a mais ampla variedade de características e capacidades, para alcançar um objetivo específico em um contexto de uso específico [ISO 2008].

Pesquisas têm sido realizadas sobre a implementação de sistemas computacionais para melhorar a vida das pessoas com deficiências [Carvalho et al. 2016, Gomes et al. 2018, Vendome et al. 2019, Alshayban et al. 2020, Rieger et al. 2020]. É essencial proporcionar a essas pessoas o acesso a recursos tecnológicos para garantir que possam realizar suas tarefas e utilizar os sistemas da mesma forma que todos os outros usuários [Mankoff et al. 2019]. No desenvolvimento de software, a acessibilidade

é um aspecto vital para alcançar uma ampla quantidade de usuários para um produto de software, e em alguns casos, essa é uma obrigação legal [Paiva et al. 2021].

Os desenvolvedores de software desempenham um papel crucial na promoção da acessibilidade digital, mas muitos enfrentam desafios ao criar aplicativos mais acessíveis. Entre os argumentos citados para o não emprego da acessibilidade se destacam: a falta de conhecimento, tempo do projeto, ausência de treinamento, não definição de requisitos claros e o desinteresse das diferentes partes interessadas [Leite et al. 2021, Gomes et al. 2020]. Além disso, recursos limitados ou inadequados, incluindo métodos de desenvolvimento e ferramentas de suporte, bem como a escassez de especialistas [Bi et al. 2022], também podem afetar a implementação da acessibilidade digital.

Estudos sugerem que a adoção de práticas ágeis de desenvolvimento pode aumentar o engajamento dos desenvolvedores [Oliveira e França 2019, Cruz et al. 2019]. Propostas integrando a usabilidade com métodos do desenvolvimento ágil têm sido consideradas [Losada et al. 2013, Hess et al. 2013, Salman e Deraman 2022], por exemplo, combinando ciclos curtos de entrega, ou definindo a usabilidade de aplicativos com desenvolvimento orientado a testes [Wolkerstorfer et al. 2008].

Esse contexto impulsiona estudos baseados em evidências científicas, explorando o potencial de abordagens teóricas e práticas para avançar o conhecimento e buscar soluções no desenvolvimento de aplicativos móveis acessíveis, visando aprimorar o estado da arte nessa área. Este trabalho está alinhado ao segundo desafio pontuado nos Grandes Desafios de Pesquisa em Interação Humano-Computador no Brasil (I GranDIHC-BR): Acessibilidade e Inclusão Digital [Baranauskas et al. 2012]. O desafio 2 advoga que é *importante a construção de interfaces acessíveis, flexíveis e ajustáveis considerando os variados contextos e a diversidade cultural brasileira*. Além disso, está alinhado com o desafio GC3: Pluralidade e Decolonialidade em IHC, do GrandIHC-BR 2025-2035 [de Oliveira et al. 2024].

Este artigo traz o relato da experiência dos autores deste trabalho nos estudos sobre interfaces móveis acessíveis. Todos os autores são especialistas em acessibilidade e atuam há diversos anos na indústria de software ou na academia. Foram, portanto, testemunhas dos avanços no tema, mas também são cientes de que há diversos desafios a serem superados. Espera-se assim, que este relato traga luz ao panorama atual sobre acessibilidade em dispositivos móveis, destacando o que existe na literatura e informando caminhos para fomentar o desenvolvimento acessível por parte dos desenvolvedores.

Como metodologia, esta pesquisa adotou uma abordagem exploratória e iterativa, estruturada em etapas interdependentes. Primeiramente, foi realizada uma revisão da literatura para identificar os principais desafios da acessibilidade em aplicativos Android, mapear ferramentas existentes e investigar como o tema tem sido tratado no processo de desenvolvimento. A partir dessas análises, foram definidos critérios de teste e estratégias para apoiar desenvolvedores. Essas estratégias foram então discutidas em entrevistas com profissionais da indústria, cujos feedbacks — aliados às críticas de revisores em submissões anteriores — permitiram realinhar e refinar a proposta. Como resultado desse processo reflexivo, foram consolidados dois principais entregáveis: (i) um conjunto de critérios de teste para problemas recorrentes de acessibilidade, inspirados na estrutura da WCAG, e (ii) um Kit de Testes Locais de Acessibilidade (AATK), voltado

à automatização da verificação de requisitos em componentes amplamente utilizados no desenvolvimento Android.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os conceitos fundamentais que embasam este trabalho. Em seguida, a Seção 3 descreve a experiência dos autores na combinação de técnicas e ferramentas existentes com o suporte teórico, processo que culminou na elaboração das principais contribuições da pesquisa. Por fim, a Seção 4 discute os principais achados e apresenta as considerações finais.

2. Referencial Teórico

Parte dos esforços da literatura sobre acessibilidade têm sido direcionados para o desenvolvimento e organização de diretrizes e recomendações. Como guia para verificar a conformidade em um contexto móvel, a *Web Accessibility Initiative* (WAI) publicou o documento "*Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile*" [W3C 2022] em fevereiro de 2015. Esse documento descreve como os princípios, diretrizes e critérios de sucesso do WCAG podem ser aplicados a dispositivos móveis. A WAI havia publicado anteriormente outros dois documentos (*Mobile Web Best Practices* em 2008 e *Mobile Web Application Best Practices* em 2010) [W3C 2022], que contêm um conjunto de melhores práticas para fornecer conteúdo Web para dispositivos móveis e desenvolver aplicativos web ricos e dinâmicos para dispositivos móveis, respectivamente.

Os desenvolvedores de aplicativos móveis também podem recorrer às diretrizes de usabilidade dos fabricantes para orientação. Documentos com essas diretrizes estão disponíveis para diversas das grandes empresas: Google [Google 2023a], Apple [Apple 2022] e Samsung [Developers 2022], por exemplo. *Frameworks* e modelos arquiteturais também podem orientar e acelerar o desenvolvimento de aplicativos acessíveis [de Oliveira e Filgueiras 2018].

Outras iniciativas, como as Normas e Diretrizes de Acessibilidade para Dispositivos Móveis da BBC (*British Broadcasting Corporation*) [BBC 2022] e o GuAMA (Guia para o Desenvolvimento de Aplicações Móveis Acessíveis) [Siebra et al. 2017, Samsung Instituto de Desenvolvimento para Informática (SIDI) 2017], também promovem a organização de diretrizes de acessibilidade para aplicativos móveis. Esses documentos incluem um conjunto de melhores práticas, recomendações e padrões para aplicativos nativos, Web ou híbridos, além de recomendações para designers de UX/UI (UX - *User Experience*, UI - *User Interface*), melhores práticas para desenvolvedores, incluindo exemplos de implementação, e estratégias para planejamento e teste de acessibilidade de aplicativos. No entanto, apesar dos esforços na literatura, problemas de acessibilidade ainda são encontrados nos aplicativos mais populares disponíveis nas lojas de aplicativos móveis [Yan e Ramachandran 2019]. A seção a seguir descreve os problemas de acessibilidade mais recorrentes em interfaces para dispositivos móveis.

2.1. Problemas de Acessibilidade

O número de aplicativos diferentes disponíveis na *Google Play Store* de dezembro de 2009 a março de 2022 atingiu cerca de 3,5 milhões[1]. Yan e Ramachandran [Yan e Ramachandran 2019] avaliaram 479 aplicativos Android na *Google Play* e identificaram que 94,8% deles tinham violações de acessibilidade, e 97,5% tinham potenciais violações de acessibilidade. O estudo identificou cinco categorias de *widgets* presentes em 92% do total de elementos de interface (são eles: *TextView*, *ImageView*, *View*, *Button* e *ImageButton*). Esses *widgets* causaram 89% das violações e 78% das potenciais violações nas aplicações avaliadas. As violações foram principalmente causadas pela falta de foco nos elementos, pela descrição ausente dos elementos, o baixo contraste de cores do texto, a falta de espaçamento suficiente entre os elementos e tamanhos de fontes de texto abaixo do mínimo necessário [Yan e Ramachandran 2019].

Dias et al. (2021) organizaram os principais problemas de acessibilidade com os quais os usuários com deficiência visual frequentemente se deparam ao interagir com os aplicativos móveis [Dias et al. 2021] . Esses problemas são listados na Tabela 1. Os autores mapeiam os problemas para um conjunto de 25 recomendações de acessibilidade móvel (mARs), adquiridas da literatura como possíveis soluções para reduzir as barreiras de acessibilidade.

2.2. Acessibilidade e o Processo de Desenvolvimento

Com o objetivo de identificar métodos e processos para o desenvolvimento de aplicativos móveis acessíveis, os autores realizaram um levantamento exploratório da literatura, com inspiração nos princípios de um Mapeamento Sistemático [Kitchenham et al. 2010]. A estratégia de busca adotada considerou o uso do termo "usabilidade" como sinônimo de "acessibilidade", devido a conflitos e intersecções identificados na literatura [Thatcher et al. 2002, Petrie e Bevan 2009] e para estabelecer um mapa mais abrangente das abordagens utilizadas integrando Interface Humano-Computador (IHC) e Engenharia de Software.

O principal interesse desse levantamento foi identificar abordagens aplicadas ao desenvolvimento *mobile* que contemplassem aspectos de acessibilidade e usabilidade. Embora os procedimentos completos da busca e análise não sejam apresentados neste artigo, os achados forneceram subsídios para a compreensão do cenário e fundamentaram os tópicos discutidos a seguir.

A maior parte dos estudos selecionados apresenta abordagens envolvendo Design Centrado no Usuário (UCD - *User-Centered Design*), com apoio às etapas de análise de requisitos e projeto arquitetural do software. Outras abordagens recorrentes são o Desenvolvimento Dirigido a Modelos (MDD - *Model-driven Development*), a Engenharia de Requisitos, Análise de Tarefas, Métodos Baseado em Cenários e *Guidelines*. Foram identificados poucos estudos com abordagens que apoiem a etapa de desenvolvimento de software.

Um exemplo de abordagem de desenvolvimento de software encontrado na literatura é o **Desenvolvimento Ágil de Software** (ASD - *Agile Software*

[1]Disponível em: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

Tabela 1. Problemas de acessibilidade mais enfrentados por pessoas com deficiência visual ao interagirem com aplicativos móveis - compilado de estudos da literatura [Dias et al. 2021].

Id.	Problemas	Referências
$\mathbf{P}_1$	Contraste de texto e/ou imagem insuficiente	[Ballantyne et al. 2018] [Acosta-Vargas et al. 2020] [Alshayban et al. 2020]
$\mathbf{P}_2$	Tamanho do alvo de toque inadequado	[Ballantyne et al. 2018] [Acosta-Vargas et al. 2020] [Alshayban et al. 2020] [Damaceno et al. 2018]
$\mathbf{P}_3$	Falta de rótulo no componente	[Carvalho et al. 2018] [Ballantyne et al. 2018] [Alshayban et al. 2020] [Damaceno et al. 2018]
$\mathbf{P}_4$	Descrição do elemento inadequada ou redundante	[Acosta-Vargas et al. 2020] [Alshayban et al. 2020] [Damaceno et al. 2018]
$\mathbf{P}_5$	*Feedback* inadequado ou indisponível	[Carvalho et al. 2018] [Ballantyne et al. 2018] [Damaceno et al. 2018]
$\mathbf{P}_6$	Imagens e/ou ícones sem alternativa textual	[Carvalho et al. 2018] [Ballantyne et al. 2018]
$\mathbf{P}_7$	Problemas e/ou limitação da Tecnologia Assistiva	[Carvalho et al. 2018] [Ballantyne et al. 2018] [Damaceno et al. 2018]
$\mathbf{P}_8$	Elementos de UI relacionados não agrupados	[Alshayban et al. 2020]
$\mathbf{P}_9$	Funcionalidade inexistente	[Alshayban et al. 2020]
$\mathbf{P}_{10}$	Organização de conteúdo inconsistente ou confuso	[Carvalho et al. 2018]
$\mathbf{P}_{11}$	Interação e/ou navegação confusa	[Carvalho et al. 2018] [Damaceno et al. 2018]
$\mathbf{P}_{12}$	Apresentação padrão não adequada	[Carvalho et al. 2018]
$\mathbf{P}_{13}$	Estrutura de formulário com *layout* confuso	[Ballantyne et al. 2018]

Development). Embora os estudos que trazem a acessibilidade ainda sejam exploratórios [Pellegrini et al. 2020], a integração da usabilidade com o ASD e alinhado ao UCD, tem aumentado [Curcio et al. 2019]. Um conjunto de estudos podem ser encontrados neste sentido [Losada et al. 2013, Hess et al. 2013, Salman e Deraman 2022].

Losada et al. (2013) apresentam uma proposta de integração de técnicas de engenharia de usabilidade com o processo de desenvolvimento ágil [Losada et al. 2013]. No trabalho é utilizada uma metodologia nomeada de *InterMod* para avaliação de usabilidade desde as fases iniciais do desenvolvimento. Os autores apresentam três principais estratégias no processo: uso de protótipos de papel, avaliação contínua da usabilidade por meio de heurísticas e análise de *logs* de interação do usuário.

Wolkerstorfer et al. (2008) propõem um "processo de usabilidade ágil", no qual combinam as vantagens da metodologia ***EXtreming Programming* (XP)**, como

ciclos curtos de entrega e estreita integração com o cliente, com instrumentos de usabilidade inspirados em práticas de UCD, como Personas, Estudos de Usuário, Testes de Usabilidade e Avaliações Heurísticas de Usabilidade [Wolkerstorfer et al. 2008]. Hess et al. (2013), por sua vez, descrevem um método centrado no usuário, baseado em um *framework* de requisitos orientados a tarefa, chamado *mConcAppt* [Hess et al. 2013]. O método é similar ao processo **Scrum** e provê conceitos de interação testáveis após cada iteração.

Os métodos ágeis surgiram em resposta à necessidade de melhorar o desenvolvimento de software, enfatizando flexibilidade e interação em vez de planejamento e documentação prévios. A adoção de métodos ágeis incentiva a interação com o cliente, motiva os desenvolvedores e torna o processo dinâmico, o que favorece entregas mais rápidas e maior qualidade [Beck et al. 2001]. Essas características podem favorecer o diálogo com diferentes partes interessadas sobre o requisito de acessibilidade.

Sob outra perspectiva, o teste de software é uma técnica importante para escrever software de alta qualidade [Crispin e Gregory 2009, Delamaro et al. 2016, Winkelmann et al. 2022]. Uma forma de permitir testes constantes e contínuos em todo o software é usando testes automatizados [Aniche 2014]. A duas próximas seções destacam os testes de software automatizados para plataformas móveis e testes existentes para analisar acessibilidade em aplicativos móveis.

2.3. 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 [Crispin e Gregory 2009]. 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 [Delamaro et al. 2016].

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 [Google 2023a].

Uma abordagem recomendada para testes é ilustrada na Figura 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 [Aniche 2014, Google 2023a, Crispin e Gregory 2009].

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

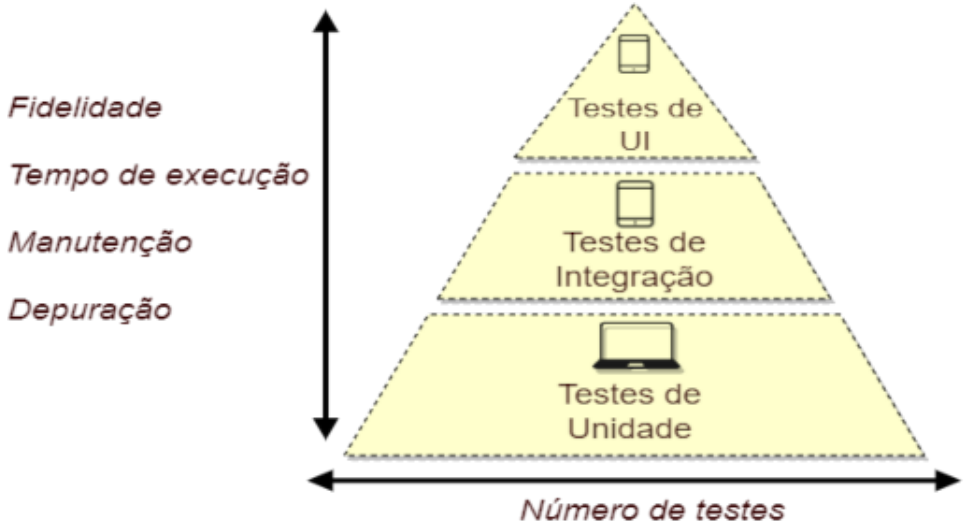

Figura 1. Pirâmide de testes representando os níveis de teste de software: da base ao topo, testes de unidade, integração e interface de usuário. [Google 2023a].

do usuário [Google 2023a]. Testes de unidade instrumentados são possíveis, porém, são significativamente mais lentos do que os testes de unidade locais [Google 2023a, Aniche 2014].

Verificações de acessibilidade em aplicativos móveis podem ser facilitadas com a utilização de ferramentas, mas para se ter maior garantia que os aplicativos sejam projetados de acordo com as diretrizes já mencionadas – como a WCAG, é preciso combinar várias abordagens.

No contexto de aplicativos Android, a própria documentação para desenvolvedores da plataforma sugere quatro abordagens:

- **Teste manual**. Os testes manuais exploram o aplicativo da forma como o usuário o utilizará. Podem ser realizados, por exemplo, interagindo com o aplicativo utilizando os serviços de acessibilidade do Android, tais como o "*TalkBack*"[2], o "*BrailleBack*"[3] e o "*Voice Access*"[4];
- **Teste com ferramentas de análise**. Favorecem a identificação de problemas por meio do código e interface;
- **Teste automatizado**. Dentro da Plataforma Android, há suporte para diversos *frameworks* de teste que possibilitam a criação e execução de testes automatizados para avaliar a acessibilidade do aplicativo;
- **Teste de usuários**. *Feedback* de usuários alvo que interagem com o aplicativo após o seu desenvolvimento ou por meio de protótipos.

[2] *TalkBack* é software leitor de tela para smartphones Android. Trata-se de um recurso de narração que ajuda pessoas com deficiência visual a usarem as funções do aparelho e que pode ser ativada nas configurações do dispositivo.

[3] *BrailleBack* é um serviço de acessibilidade que ajuda usuários cegos a usar dispositivos Braille, e funciona em conjunto com o *TalkBack*.

[4] *Voice Access* é um serviço de acessibilidade que ajuda usuários que têm dificuldade em manipular tela de toque, pois permite controlar todo o sistema com a voz.

A plataforma Android disponibiliza um conjunto de ferramentas que podem ser utilizadas nas diferentes estratégias de testes de acessibilidade. As verificações de acessibilidade pré-definidas nessas ferramentas são baseadas na biblioteca ***Accessibility Test Framework*** (ATF). A ATF é uma biblioteca Android para avaliar a acessibilidade de aplicativos que coleta verificações relacionadas à acessibilidade em *Views* e objetos *AccessibilityNodeInfo* (que o Android deriva das *Views* e envia para *AccessibilityServices*) [Google 2022]. A ATF possui um conjunto pré-definido de checagens de acessibilidade, e fornece suporte/recursos para execução ou personalização das verificações de acessibilidade. Dentre esses, uma classe para habilitar verificações de acessibilidade automatizadas [Google 2023a].

Embora seja uma biblioteca aberta, que pode ser utilizada por desenvolvedores com outros testes e ferramentas, há pouca documentação disponível. As verificações de acessibilidade atualmente disponíveis no ATF incluem presença de labels, tamanho da área de toque, contraste de cores e outras propriedades de elementos de UI relacionados com serviços de acessibilidade do Android.

2.4. Ferramentas para Testes de Acessibilidade no Android

Visando as soluções que possam estar mais integradas ao ambiente de desenvolvimento, foram investigadas com mais detalhes as ferramentas disponíveis para testes com ferramenta de análise e testes automatizados.

2.4.1. Android Lint

O Lint [Google 2023a] é uma ferramenta de verificação de código do Android Studio[5] que ajuda a identificar e corrigir problemas estruturais no código-fonte. Ele reporta cada problema com uma mensagem descritiva e um nível de gravidade, permitindo que os desenvolvedores priorizem as correções necessárias. O Lint verifica arquivos de origem do projeto Android em critérios de precisão, segurança, desempenho, usabilidade, acessibilidade e internacionalização.

As inspeções de acessibilidade realizadas pelo Lint podem ser consultadas no próprio Ambiente de Desenvolvimento Integrado (IDE - *Integrated Development Environment*) do Android Studio, e contemplam acessibilidade em *Views* personalizadas, imagem sem descrição, *widget* inacessível por teclado, rótulo de acessibilidade ausente ou sobrescrita do método *getContentDescription()* em uma *View*.

2.4.2. Scanner de Acessibilidade

O Scanner de Acessibilidade [Google 2023a] é um aplicativo disponibilizado pelo Google que, quando habilitado, é capaz de analisar a tela de qualquer aplicativo e apresentar ao usuário uma visão que destaca os problemas de acessibilidade diretamente na interface de usuário. Esses problemas destacados podem ser clicados para obter informações mais detalhadas e sugestões. As verificações de acessibilidade realizadas pelo *app* são baseadas na ATF e incluem conteúdo dos rótulos (rótulo de item ausente, rótulos de item

[5]Disponível em: `https://developer.android.com/studio`

duplicados, campos de entrada com descrições de conteúdo e rótulos com informações redundantes), itens clicáveis (subconjuntos clicáveis de itens de texto e limites clicáveis duplicados), contraste e tamanho da área de toque.

2.4.3. *UI Automator*

O *UI Automator* [Google 2023a] é um *framework* de teste fornecido pela plataforma Android para testar a Interface do Usuário (do inglês, *User Interface* - UI) de aplicativos Android. O *UI Automator* permite que os desenvolvedores escrevam testes funcionais automatizados para seus aplicativos Android, que simulam a interação do usuário com os componentes da interface.

O *UI Automator Viewer*[Google 2023a], por sua vez, é uma ferramenta que permite aos desenvolvedores inspecionar os componentes da interface do usuário de um aplicativo Android e gerar o código de teste necessário para interagir com eles. É particularmente útil para desenvolvedores que estão criando testes automatizados usando a estrutura de teste *UI Automator*.

2.4.4. *Espresso*

O *Espresso*[Google 2023a] é um *framework* de teste de interface do usuário para a plataforma Android. Ele funciona de modo integrado ao Android Studio e é executado em um emulador ou dispositivo Android real. O Espresso é projetado para simplificar o processo de escrita de testes de UI para aplicativos Android, permitindo que os desenvolvedores criem testes automatizados do tipo estrutural (caixa branca), ou seja, testes que exploram a lógica interna e estrutura do código. Diferente dos testes funcionais (caixa preta), que se concentram nas entradas e saídas do sistema [Delamaro et al. 2016].

O *Espresso* permite que os testes interajam com os elementos da UI do aplicativo, como botões, caixas de texto e menus de opções, além de realizar ações como clicar em botões e inserir texto, entre outras. Ele também pode verificar o estado dos elementos, como a presença de texto em uma caixa de texto ou a visibilidade de um botão.

Embora seja uma ferramenta para automação de testes de interface, o *Espresso* pode ajudar a testar acessibilidade por meio da integração com a ATF e a API (*Application Programming Interface*) *AccessibilityChecks*. Em cada caso de teste, o desenvolvedor pode ativar os verificadores de acessibilidade para identificar as exceções quando qualquer componente ativado por um teste falha em qualquer das checagens de acessibilidade contempladas pela ATF.

É possível com o *Espresso*, habilitar que as checagens sejam realizadas em toda a hierarquia de *views* de uma tela. Mas, é necessário que as verificações de acessibilidade sejam ativadas em cada tela, individualmente.

2.4.5. *Roboletric*

Robolectric[Google 2023b] é um *framework* que permite escrever testes de unidade e executá-los em uma máquina virtual Java (JVM - Java Virtual Machine), enquanto usa a API do Android, ou seja, é uma estrutura de teste de unidade, que pode também testar a interação do usuário. Enquanto os testes do *Espresso* são executados usando um emulador do dispositivo Android, testes *Robolectric* são executados diretamente na máquina virtual Java e invoca diretamente a API do aplicativo. Isso torna a execução do teste mais rápida em comparação com *frameworks* em que os testes são executados em um dispositivo real ou emulador. O *Roboletric* é capaz de fornecer algum controle sobre o comportamento do *framework* do Android e permite simular recursos e métodos nativos.

O suporte para gráficos nativos do Android, no entanto, só foi adicionado ao *Robolectric* em sua versão 4.10, de abril de 2023. O suporte não é habilitado por padrão, porém, quando habilitado, as interações com as classes de gráficos do Android usam o código de gráficos nativo real do Android e possuem uma fidelidade maior [Google 2023b].

Há ainda na literatura propostas como a ferramenta *Mobile Accessibility Testing* (MATE) [Eler et al. 2018], que examina automaticamente aplicativos por meio de vários testes de acessibilidade e trata questões relacionadas a deficiências visuais e motoras. O MATE testa a acessibilidade dos elementos de UI e relata o número de falhas relacionadas a cada tipo de teste realizado, bem como disponibiliza um relatório detalhado de todas as falhas de acessibilidade dos *widgets* de UI. As propriedades de acessibilidade testadas incluem texto pronunciável/falado, tamanho da área de toque, relação de contraste, limiares de clique duplicado e extensão de clique.

Vontell (2019) desenvolveu o *Bility*, um *framework* de testes de acessibilidade de fácil utilização que inclui várias funcionalidades: uma linguagem independente de plataforma para descrever interfaces de usuário, um processo automatizado para iniciar, navegar e explorar o espaço de estados de um aplicativo Android, e um processo para testar questões dinâmicas de acessibilidade, como navegação intuitiva e por teclado no contexto do aplicativo, construindo uma máquina de estados que representa o aplicativo [Vontell 2019]. O autor introduziu novas técnicas de análise de interface de usuário, como determinar mudanças de contexto com base apenas nas informações dessa interface e navegar automaticamente em um aplicativo com pouca ou nenhuma entrada do desenvolvedor. Dessa forma, o *Bility* pode detectar problemas estáticos adicionais e descobrir problemas dinâmicos de acessibilidade não detectados por ferramentas atuais.

Outra abordagem é o *AccessibiLint* [de Oliveira e Filgueiras 2019], que estende o *Android Lint* para aumentar o escopo de regras de verificação de código estático. Os proponentes adicionaram 17 regras com base em recomendações de acessibilidade para deficiência visual. O recurso permite que o desenvolvedor faça uma revisão antecipada de acessibilidade na fase de teste, durante o desenvolvimento de aplicativos nativos do Android.

2.5. Discussão sobre as Ferramentas Identificadas

As ferramentas disponíveis para Android realizam as verificações de acessibilidade principalmente por meio da biblioteca ATF. Entre essas ferramentas, o Scanner de

Acessibilidade se destaca como uma opção de uso mais simples. No entanto, uma desvantagem dessa abordagem é que o desenvolvedor precisa ativar a ferramenta em cada tela do aplicativo para obter os resultados desejados. Essa ação adicional implica na necessidade de explorar manualmente o aplicativo, o que pode não ser escalável para aplicativos maiores ou para testes frequentes.

Tanto o *UI Automator* quanto o *Espresso* são ferramentas utilizadas para automatizar testes. A principal diferença entre elas está no tipo de teste que cada um realiza. O *UI Automator* realiza testes caixa preta, o que significa que o testador não tem acesso à estrutura interna do código do aplicativo. Por outro lado, o *Espresso* realiza testes caixa branca, o que permite uma maior visibilidade e identificação da origem dos problemas.

Ambas as ferramentas são amplamente utilizadas por testadores. O *UI Automator* é a base para outras ferramentas como o MATE e o *Appium*[6]. No entanto, é importante destacar que, com exceção do *Espresso* – que pode realizar testes intermediários, todas essas ferramentas realizam testes de UI, o que pode resultar em um *feedback* mais distante para os desenvolvedores.

Com o *Espresso*, os desenvolvedores têm a possibilidade de criar testes de acessibilidade de forma simples, pelo uso da API *AccessibilityChecks*, dentro do projeto de testes instrumentados. No entanto, é importante destacar que esses testes exigem o uso de um dispositivo físico ou emulado, o que pode torná-los mais lentos em comparação com testes locais.

O *Robolectric*, opção para criar testes de unidade com recursos do Android, chegou a oferecer suporte à API *AccessibilityChecks*. No entanto, a partir da versão 4.5 do *Robolectric*, essa funcionalidade foi descontinuada devido a várias falhas na identificação dos mesmos problemas que o *Espresso* era capaz de identificar.

Quanto às ferramentas identificadas na literatura, embora as opções baseadas em verificação estática de código, como o *AccessibiLint* [de Oliveira e Filgueiras 2019] sejam de baixo custo e devam ser integradas ao ambiente de desenvolvimento sempre que possível, elas geralmente funcionam apenas em nível sugestivo, o que significa que as melhorias de acessibilidade podem ser ignoradas ou não percebidas pelos desenvolvedores.

Em relação às ferramentas de [Eler et al. 2018] e [Vontell 2019], ainda que tenham realizado testes de acessibilidade utilizando ferramentas que auxiliam os desenvolvedores, eles priorizaram tais testes após a codificação e verificam subconjuntos dos problemas de acessibilidade definidos pelos autores. Isso indica que os testes de acessibilidade apresentam limitações em relação à abrangência e são realizados principalmente após o desenvolvimento do software, sendo essencialmente testes de UI, que são testes com *feedback* mais distante do desenvolvedor.

É importante destacar a escassez de estudos voltados para a inclusão de testes de acessibilidade durante a fase de desenvolvimento do software, especialmente desde o início da codificação. As alternativas que se integram ao ambiente de desenvolvimento, sem necessidade de execução em dispositivos Android, estão muito limitadas a avisos.

[6]http://appium.io/docs/en/2.0/

São escassas as alternativas para automatização de testes de acessibilidade com testes pequenos.

A Tabela 2 sumariza as ferramentas atualmente disponíveis para a realização de testes de acessibilidade no Android, destacando a quais testes cada uma delas se destina, e qual o esforço requerido do desenvolvedor ou testador para as verificações de acessibilidade.

Tabela 2. Ferramentas para testar acessibilidade no Android.

Ferramenta	Teste	Esforço requerido
Lint	Estático Local	Inspeção automática em conjunto pré-definido de verificações de acessibilidade.
Roboletric	Teste de Unidade Local	Escrita de testes para cada verificação de acessibilidade.
Espresso	Testes integrados Testes de UI Instrumentado	Chamada para API *AccessibilityChecks* a partir de *View* raíz de cada tela
UI Automator	Testes de UI Instrumentado	Automatização de testes de UI. Pode chamar o *AccessibilityChecks*
Scanner de Acessibilidade	Teste de UI	Execução do app em cada tela

Atualmente, não há soluções que ofereçam facilidade para testar acessibilidade com testes locais. Isso significa que os desenvolvedores precisam criar seus próprios testes de acessibilidade individualmente para ter um conjunto mínimo de testes automatizados. No entanto, muitas equipes enfrentam desafios de tempo, recursos e apoio organizacional [Swallow et al. 2016] para dedicar-se à escrita desses testes. Esses achados evidenciam a necessidade de oferecer suporte mais direto aos desenvolvedores por meio de soluções pragmáticas, o que motivou a formulação de estratégias específicas para apoiar a incorporação da acessibilidade no desenvolvimento Android.

Neste contexto, na próxima seção são descritas as experiências dos autores na exploração de técnicas e ferramentas existentes para a elaboração dessas estratégias, com foco em uma participação mais ativa dos desenvolvedores no processo.

3. Experiências explorando estratégias para desenvolvedores

Com o objetivo de aumentar o envolvimento dos desenvolvedores na produção de aplicativos móveis acessíveis, os autores exploraram combinações de conhecimento prévio da literatura com práticas comuns aos desenvolvedores, como as práticas de desenvolvimento ágil de software. Os estudos buscaram identificar uma abordagem de desenvolvimento de aplicativos móveis acessíveis que seja facilmente incorporada às atividades rotineiras dos desenvolvedores e amplamente divulgada, reduzindo as barreiras para sua adoção.

Os autores direcionaram sua exploração para responder a quatro perguntas cruciais relacionadas à estratégia de testes de acessibilidade: 1) O que deve ser testado? 2) Quais ferramentas podem ser utilizadas e em que momento? 3) Quem deve ser responsável pela realização dos testes? E 4) Como os testes devem ser executados?

As próximas subseções discutem cada uma dessas questões e, na Subseção 3.5, serão relatados e discutidos os *feedbacks* obtidos de alguns profissionais da

industria convidados para participar de uma entrevista semiestruturada sobre as estratégias selecionadas.

3.1. Definição de escopo das verificações de acessibilidade

Testar todos os problemas de acessibilidade, em todos os componentes e estruturas de interface pode alocar muito tempo dos desenvolvedores, que tendem a não escrever testes de acessibilidade dedicados. Assim, as soluções existentes podem atender apenas a um subconjunto restrito de problemas de acessibilidade [Eler et al. 2018, Vontell 2019].

Tentando entender como mitigar esse problema, os autores deste trabalho investigaram estudos prévios da literatura para analisar e sugerir um escopo de cobertura de testes de acessibilidade. O intuito era identificar os principais problemas de acessibilidade enfrentados por **usuários com deficiência visual**, tendo por base os problemas compilados por Dias et al. (2021), e indicando as respectivas recomendações para evitar tais problemas [Dias et al. 2021].

Considerou-se também, os elementos de UI em que são encontrados a maioria das violações de acessibilidade nos aplicativos mais populares de acordo [Yan e Ramachandran 2019].

A Tabela 3 descreve a relação dos problemas de acessibilidade mais recorrentes, segundo o estudo de [Yan e Ramachandran 2019] e, para cada um desses problemas, quais são as respectivas recomendações de acessibilidade, nomeadas como *maRs* (*mobile accessibility recomendations*) na compilação de Dias et al., bem como quais são os *widgets* no Android (dentre os cinco mais recorrentes de acordo com Yan e Ramachandran), que estão associados a esses problemas de acessibilidade.

Tabela 3. Problemas de acessibilidade e recomendações para os *widgets* mais usados.

Problema de Acessibilidade	Id.	maR	*Widgets*
Falta de foco do elemento	P_{11}	maR13	*EditText*, *Button*, *ImageButton*
Descrição do elemento ausente	P_3|P_6	maR10 maR01	*ImageView*, *Button*, *ImageButton*, *EditText*
Baixo contraste de cor de texto	P_1	maR07	*TextView*, *Button*, *ImageButton*, *EditText*
Falta de espaçamento suficiente entre os elementos	P_2	maR18	Layout

A experiência dos autores sugere que grande parte da dificuldade na criação de testes automatizados está torná-los genéricos e abrangentes o suficiente. Porém, ao priorizar quais testes e em quais elementos realizá-los, são estabelecidos critérios mais objetivos, com potencial de solucionar a maior parte dos problemas encontrados em recorrência.

Definido o escopo inicial a ser atendido, o passo seguinte é a escolha das ferramentas e estratégias de testes.

3.2. Distribuição dos testes e seleção de ferramentas

Muitas são as ferramentas disponíveis para desenvolvedores Android implementarem testes. Além das citadas na Subseção 2.4, existem bibliotecas e ferramentas de

testes automatizados como *JUnit*[7], *Mockito*[8], *MonkeyRunner*[9], *Robotium*[10], entre outras [Kochhar et al. 2015]. A utilização das ferramentas e os tipos de teste a serem realizados por cada uma, são definidos de acordo com a estratégia adotada. A documentação de testes do site *Android Developers* [Google 2023a] recomenda que a proporção siga o modelo de pirâmide da Figura 1, com 70% de testes pequenos, 20% de testes médios e 10% de testes grandes.

Uma possível composição do conjunto de testes é sugerida pelos autores deste trabalho e ilustrada na Figura 2. A composição pode incluir testes pequenos com *frameworks* para testes de unidade como *JUnit*, *Mockito* e *Roboletric*, todos eles podendo ser executados de forma não instrumentada. Para testes médios, é possível a utilização do *Espresso*, inclusive por meio de testes de unidade instrumentados, ou ainda pelo *UI Automator*, também de modo instrumentado. Já as opções para os testes grandes são mais numerosas, como o próprio *UI Automator*, o *Accessibility Scanner*, ou ferramentas como *Appium*, MATE [Eler et al. 2018] e *Bility* [Vontell 2019].

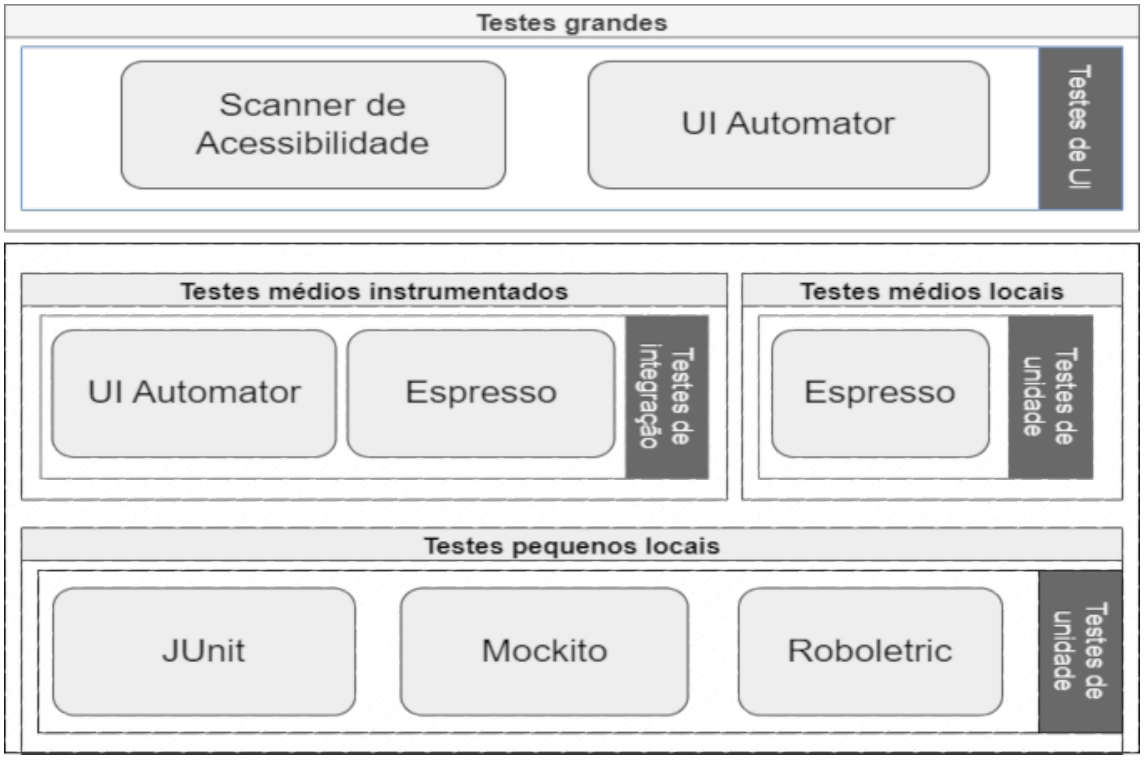

Figura 2. Composição de ferramentas para testes de acessibilidade em Android, organizadas por tipo e nível: testes grandes de UI (como o Scanner de Acessibilidade), testes médios instrumentados (com Espresso e UI Automator), testes médios locais (com Espresso) e testes pequenos locais (JUnit, Mockito e Robolectric).

São encorajadas estratégias que combinem testes de UI com testes instrumentados

[7]Disponível em: `https://junit.org/junit4/`
[8]Disponível em:`https://site.mockito.org/`
[9]Disponível em:`https://developer.android.com/studio/test/monkeyrunner?hl=pt-br`
[10]Disponível em:`https://github.com/RobotiumTech/robotium`

ou mesmo locais [Google 2023a]. No entanto, as pesquisas e a experiência dos autores mostram que os testes de acessibilidade são geralmente realizados após a codificação. Os autores acreditam que, estabelecendo responsabilidades mais claras, todos os envolvidos podem trabalhar de forma integrada para promover a acessibilidade.

3.3. Mais responsabilidade aos desenvolvedores

Durante o mapeamento sistemático realizado neste trabalho, foi observado que há poucos estudos que abordam a inclusão de acessibilidade durante a fase de desenvolvimento de software. Porém, alguns estudos que tratam de usabilidade oferecem alternativas que também podem ser desenvolvidas para evitar problemas de acessibilidade.

A abordagem proposta por Wolkerstorfer et al. (2008), por exemplo, combina práticas de XP com métodos UCD para um "processo de usabilidade ágil". Um dos instrumentos utilizados pelos autores foi o teste de unidade estendidos (*Extended unit tests*), adicionando casos de teste específicos de usabilidade [Wolkerstorfer et al. 2008]. Os autores adicionam semântica (o rótulo correto de um botão, por exemplo) aos testes baseados em código, o que permite que possam verificar o cumprimento de diretrizes (como o uso de letras maiúsculas em botões, no mesmo exemplo). A abordagem é baseada no desenvolvimento orientado a testes (TDD - *Test-driven development*), ou seja, em escrever os testes antes de desenvolver a funcionalidade. Assim, ao adicionar testes de unidade com semântica relacionados à usabilidade, os desenvolvedores conseguem definir a usabilidade da aplicação.

Seguindo o mesmo princípio, a primeira sugestão que os autores deste trabalho deixam para incentivar desenvolvedores a realizarem testes de acessibilidade em seus projetos, consiste em uma abordagem orientada a testes para o desenvolvimento de requisitos de acessibilidade, chamada **acTDD**. A acTDD tem o objetivo de auxiliar os desenvolvedores a criarem aplicativos móveis acessíveis e é fundamentada em três pilares: a transformação do conjunto de recomendações de acessibilidade em requisitos de software; a priorização dos elementos de interface que têm mais violações de acessibilidade, de acordo com a literatura; e a distribuição dos testes para que haja mais testes locais do que testes instrumentados, reduzindo o tempo de *feedback*. Com essa proposta, detalhada na seção seguinte, os desenvolvedores assumem a responsabilidade de implementar requisitos de acessibilidade como parte dos requisitos de software.

3.4. Uma proposta para inserir requisitos de acessibilidade com TDD

Durante o desenvolvimento dirigido por testes, o desenvolvedor é estimulado a criar um teste para cada novo requisito de software. Em uma abordagem de acessibilidade dirigida por testes, cada recomendação de acessibilidade a ser considerada se torna um novo requisito e requer um novo teste. Seguindo essa sugestão de fluxo de desenvolvimento, o desenvolvedor pode buscar resolver os problemas de acessibilidade que desejar, nos elementos de UI que escolher, pois terá como ponto de partida um novo requisito.

Embora essa proposta não limite a quantidade de problemas ou elementos a serem abordados, desenvolver testes de acessibilidade para uma grande gama de problemas e elementos pode não ser escalável [Eler et al. 2018]. No entanto, pelos relados extraídos literatura, é possível assumir que se o desenvolvedor priorizar os problemas e elementos elencados na Subseção 3.1, testes bem sucedidos podem evitar boa parte dos problemas mais recorrentes em aplicativos Android.

No exemplo abaixo será demonstrado como se dá o fluxo de implementação de um requisito de acessibilidade pela abordagem acTDD.

Exemplo: baixo contraste da cor do texto

O primeiro passo, no TDD, é sempre a criação de um teste para um novo requisito de software. Neste exemplo, o que se deseja é evitar o problema de acessibilidade de contraste de texto e/ou imagem insuficiente, apresentado na Tabela 1 como P1. Para esse problema, a literatura apresenta como **requisito** a recomendação de acessibilidade **maR07**.

"maR07 - Contraste de cor: deve-se utilizar uma taxa de contraste de cores pelo menos **4.5:1**, pois ajuda os usuários a identificar melhor o conteúdo do app" [Dias 2021].

Para inferir a taxa de contraste de cores de um texto, em relação ao seu *background*, é preciso acessar as cores utilizadas no texto e no *background*. Nesse exemplo, a taxa de contraste de cores foi obtida utilizando a biblioteca *ColorUtils* do Android, que implementa o cálculo do contraste, seguindo a fórmula proposta pela W3C[11].

Uma maneira de realizar essa inferência no Android é por meio de um teste instrumentado utilizando o *Espresso*[12]. Para checar a taxa de contraste de um *TextView* em tela, o teste acessa a cor do texto e do *background*, e obtém a taxa de contraste. Se o resultado for inferior a 4.5, o teste falha, e será necessário realizar alterações nas cores do elemento até que o teste passe.

Embora eficaz, a execução do teste pelo *Espresso* custa tempo de execução, pois requer que o emulador seja iniciado e a aplicação reinstalada no dispositivo, para então rodar o teste.

Uma alternativa mais adequada, de acordo com a distribuição da pirâmide de testes (ver Figura 1), seria implementar a mesma verificação como um teste local, por exemplo, com um teste utilizando o *Roboletric*[13], que não exige a execução por meio de um dispositivo físico ou emulado.

O exemplo apresentado não é um modelo definitivo, mas sim uma demonstração do fluxo de implementação com o acTDD. É importante ressaltar que o teste pode ser refatorado e aprimorado em um ciclo típico de TDD. Além disso, os desenvolvedores podem considerar a melhoria do teste, aplicando-o a todos os elementos da interface do usuário de uma tela que pertença ao subconjunto especificado. Ao finalizar um ciclo de implementação de requisito com o acTDD, o desenvolvedor tem: a) um requisito de acessibilidade atendido; e b) um teste automatizado para checar eventuais alterações nas propriedades da interface.

Cabe dizer que o TDD é uma técnica de desenvolvimento – e não de testes. Assim, a proposta do acTDD exige que os testes sejam escritos durante o desenvolvimento – e

[11]WCAG 2.0 contrast ratio - `https://www.w3.org/TR/2008/REC-WCAG20-20081211/#contrast-ratiodef`

[12]Ver exemplo no link `https://ibb.co/tH4FQTw`

[13]Ver exemplo no link `https://ibb.co/gtv5qMw`

não em etapas posteriores. O que confere mais responsabilidade aos desenvolvedores. No entanto, pela experiência dos autores, é uma proposta que exige uma mudança de paradigmas.

A fim de obter uma perspectiva pragmática, não apenas do acTDD, mas de todas as estratégias elencadas – como a delimitação do escopo, as ferramentas selecionadas e a distribuição dos testes – foram ouvidos profissionais indústria, conforme apresentado na Subseção 3.5.

3.5. Entrevistas com profissionais da indústria

Como parte do processo de construção e consolidação das estratégias elaboradas neste trabalho, foram realizadas entrevistas semiestruturadas com cinco profissionais atuantes na área de desenvolvimento de software, todos com experiência em projetos *mobile*. Os participantes foram selecionados por conveniência [Yin 2016], e os encontros ocorreram virtualmente, com base em um roteiro previamente elaborado.

O objetivo foi compreender como a acessibilidade é abordada nesses contextos, como se distribuem as responsabilidades relacionadas ao tema e obter percepções sobre uma proposta de desenvolvimento de aplicativos acessíveis orientados por testes. Uma breve apresentação do contexto da pesquisa e das estratégias em elaboração — como a delimitação de escopo para testes, as ferramentas consideradas, a distribuição de responsabilidades e a proposta de acTDD — era feita de forma oportunística, de acordo com a condução da conversa.

Esse formato permitiu que os participantes apresentassem suas percepções, experiências e sugestões de forma livre, favorecendo a coleta de conhecimento tácito. Segundo Nonaka e Takeuchi (1995), esse tipo de conhecimento é puramente pessoal, profundamente enraizado nas ações, experiências, valores e emoções, sendo difícil de formalizar e criado em um contexto prático específico [Nonaka e Takeuchi 1995, p. 66]. Tal conhecimento pode ser adquirido ao colher informações de profissionais com experiência prática e estruturas mentais como atitudes e crenças [Nonaka e Takeuchi 1995].

A Tabela 4 apresenta a equipe de trabalho dos profissionais entrevistados e suas principais experiências no escopo desta pesquisa, associadas a uma identificação que será usada nas discussões subsequentes.

Tabela 4. Perfil dos profissionais entrevistados.

Id.	Equipe de trabalho	Principais experiências
E1	Garantia da Qualidade	Automação de testes
E2	Desenvolvedor	Desenvolvimento *mobile*
E3	Garantia da Qualidade	Automação de testes para Web
E4	Garantia da Qualidade	XP e Automação de testes para *mobile*
E5	*Tech Leader*	Projetos Android e testes no Android

As respostas foram analisadas por meio da síntese das falas e da identificação de padrões e divergências entre os entrevistados, permitindo apoiar a avaliação e o refinamento das estratégias inicialmente propostas.

3.5.1. *Feedbacks* dos entrevistados

Os *feedbacks* obtidos a partir das entrevistas semiestruturadas agregaram importantes reflexões sobre essa pesquisa e complementaram a experiência desses autores. Além das opiniões dos entrevistados, foram consideradas para essa análise, os *feedbacks* recebidos na revisão de um artigo científico submetido em outubro de 2022 para o *Symposium On Applied Computing* - SAC[14]. De acordo com um dos revisores da publicação submetida, embora as diretrizes sejam pré-definidas, não é necessário que os testadores construam os testes individualmente de antemão, e o TDD pode não ser a abordagem adequada para resolver o problema apresentado. A mesma opinião foi defendida por E5, que lembra que TDD tem sido mais útil explorando conteúdo não conhecido e que *apps mobile* normalmente possuem pouca regra de negócio.

A abordagem proposta neste trabalho, que busca incentivar o uso de testes de acessibilidade por parte dos desenvolvedores de forma prática, pressupõe maior participação desses atores. No entanto, a experiência dos autores deste trabalho, aliada aos estudos da literatura, permitem informar que essa não é a maneira como a acessibilidade tem sido considerada atualmente. Durante as entrevistas, quando perguntados sobre como são feitas as verificações de acessibilidade nos projetos em que atuavam, os entrevistados indicaram que elas costumavam acontecer essencialmente na etapa de testes, ou seja, na etapa final do ciclo de desenvolvimento, e realizados por testadores, não necessariamente os desenvolvedores. E1, por exemplo, afirma que os testadores automatizam testes intermediários e de UI, e os testes automatizados são usados para acelerar testes de regressão.

O entrevistado E5 acredita que um conjunto de testes de acessibilidade entregaria mais valor com testes instrumentados. E5 entende que a pirâmide de testes não é fidedigna ao contexto *frontend*, sendo preferível uma distribuição em formato hexagonal, ou seja, com mais testes de integração, em comparação com testes de unidade e testes de UI. Opinião parecida foi apontada por um dos revisores da publicação submetida, que argumenta ser preferível que testes de acessibilidade sejam realizados por testes de UI.

Para provocar uma mudança na forma como os testes são pensados e executados pela indústria de software atualmente, é necessário discutir responsabilidades. Os autores deste trabalho advogam que essa responsabilidade deve ser dividida entre toda a equipe, incluindo designers, desenvolvedores e testadores. Esse argumento corrobora com a definição de UX cunhada por Donald Norman, que considera que a boa experiência do usuário alvo deve ser pensada por todos do time de desenvolvimento, cada um dando a sua contribuição de acordo com o seu papel no processo[15].

O que se observou com as entrevista, no entanto, é que os diferentes perfis acham que a responsabilidade deve ser de outros atores, e não necessariamente do seu perfil. E1 entende que faz sentido antecipar a implementação de requisitos de acessibilidade, pois quanto mais cedo resolver o requisito, menor o custo de desenvolvimento. Esse entendimento pressupõe que a responsabilidade seja do designer. E2, o entrevistado com o perfil de desenvolvedor e, portanto, presente na etapa de desenvolvimento, entende que

[14]Disponível em: `https://www.sigapp.org/sac/sac2023/`

[15]Disponível em: `https://www.nngroup.com/articles/definition-user-experience/`

uma abordagem TDD entrega muita responsabilidade ao desenvolvedor, e que, mesmo com um conjunto definido de problemas e elementos a serem verificados, esses atores podem ter dificuldade para implementar soluções generalistas. Ao mesmo tempo, E2 entende que o desenvolvimento de uma ferramenta que entregue soluções prontas para este fim, ainda que possível, demandaria um trabalho grande para ser desenvolvida. Para E2, essa atribuição – garantir a acessibilidade dos *apps* – deve ser dos *designers* de UI/UX. E5 corrobora com essa opinião e declara que um bom *Design System* já contempla requisitos de acessibilidade.

Quanto ao suporte recebido para lidar com questões de acessibilidade, E1 afirma que não é dada prioridade à alocação de tempo para implementar testes de acessibilidade. Já E5 relata que em todas as empresas e projetos em que trabalhou, a acessibilidade tem sido tratada apenas como uma estratégia comercial.

A partir da análise das falas, foram identificados padrões de percepção e barreiras recorrentes entre os profissionais, revelando pontos críticos para a adoção de práticas acessíveis durante o desenvolvimento. A Tabela 5 apresenta uma síntese das principais percepções extraídas das entrevistas, organizadas por aspecto investigado.

Tabela 5. Síntese das percepções e padrões observados a partir das entrevistas

Aspecto	Percepções e padrões observados
Verificação de acessibilidade	Predominantemente realizada por testadores, na fase final do ciclo
Ferramentas utilizadas	Testes instrumentados ou manuais; pouca presença de testes locais automatizados
Responsabilidade pela acessibilidade	Desenvolvedores consideram que designers deveriam ser os principais responsáveis
Barreiras identificadas	Falta de tempo, apoio institucional e recursos prontos para adoção

Esses insights, somados às críticas recebidas em avaliações anteriores, contribuíram diretamente para o aprimoramento das estratégias inicialmente propostas, culminando na consolidação dos principais artefatos desta pesquisa, conforme discutido a seguir.

4. Discussões e Considerações finais

Os argumentos contrários à proposta baseada em TDD são relevantes. Partem de um padrão estabelecido na literatura: testes de acessibilidade são essencialmente testes de UI, pois testam requisitos não funcionais. Essa prática, no entanto, não tem sido suficiente para evitar os problemas mais triviais e recorrentes de acessibilidade em aplicativos móveis [Yan e Ramachandran 2019].

De modo geral, requisitos funcionais descrevem o que o software deve fazer, enquanto requisitos não funcionais especificam atributos de qualidade, como usabilidade, confiabilidade, eficiência, manutenibilidade e portabilidade [Turine e Masiero 1996]. Com frequência, esses requisitos não funcionais são documentados de forma separada, com descrições vagas ou sem critérios mensuráveis, o que dificulta sua análise e teste adequados [Eckhardt et al. 2016].

Segundo Eckhardt et al. (2016), muitos dos chamados requisitos "não funcionais" descrevem comportamentos observáveis do sistema e, portanto, deveriam ser tratados

como funcionais. Essa mudança de perspectiva ajudaria a incluir tais requisitos em ciclos de teste e desenvolvimento de forma mais integrada.

Existem desafios técnicos importantes na criação de testes de acessibilidade pelo desenvolvedor. Nem todo requisito pode ser verificado sem o uso de um dispositivo físico ou emulado. Além disso, os testes manuais e os testes com usuários permanecem insubstituíveis. No entanto, é possível — e desejável — automatizar testes para os problemas de acessibilidade mais recorrentes, especialmente nos componentes mais utilizados. Quando integrados aos ciclos de integração e entrega contínua (CI/CD), esses testes podem contribuir para a entrega de aplicativos mais acessíveis.

Nesse sentido, embora o TDD não seja necessariamente a solução ideal para esse cenário, recomenda-se incentivar o uso de testes locais de acessibilidade, integrados ao ambiente de desenvolvimento. A participação dos desenvolvedores deve ser promovida desde as fases iniciais do projeto, atuando de forma complementar às demais funções da equipe.

O uso dessa estratégia não significa, no entanto, que as responsabilidades dos outros papéis devam ser diminuídas. Ao contrário, o envolvimento de todos e a auto-organização das equipes fazem parte dos princípios do desenvolvimento ágil [Beck et al. 2001, Pressman 2011].

Independentemente de quem assuma a maior responsabilidade, **é imperativo que a inserção de requisitos de acessibilidade seja priorizada**. Garantir que todas as pessoas possam usar os recursos tecnológicos, inclusive aquelas com deficiência, é uma obrigação legal [BRASIL 2015]. A acessibilidade não deve mais ser tratada como um diferencial opcional, ou algo "*nice to have*".

Trata-se de um requisito amplamente discutido na academia, mas ainda pouco efetivo na prática da indústria. Este trabalho busca contribuir com esse cenário, não apenas ao retomar o tema, mas também ao apresentar recomendações concretas para fomentar sua implementação e testes no contexto real de desenvolvimento.

Entende-se ainda, que é preciso um amplo processo de formação dos profissionais da área de desenvolvimento de software com o olhar sobre acessibilidade. O que se percebe é que esse não é um requisito devidamente estudado pelos alunos de graduação nas áreas de TI durante a sua formação, então, quando deparados com a necessidade, entendem que deve ser um requisito a mais, com recurso financeiro extra envolvido e/ou com alocação muito maior de tempo do projeto. Nesse quesito, academia precisa se aproximar mais da indústria, não só na formação de profissionais mais bem preparados, como também na proposição de soluções que sejam efetivas.

As seção a seguir destaca algumas possíveis estratégias, considerando o que foi observado com a revisão de literatura, o estudo das ferramentas de mercado, a entrevista com profissionais, além da experiência dos autores no tema.

4.1. Possíveis estratégias

A proposta de incluir requisitos de acessibilidade em um ciclo semelhante ao do TDD é relevante, embora necessite de mais investigação. As premissas que sustentaram essa proposta continuam válidas, incluindo a distribuição e delimitação de responsabilidades, maior envolvimento dos desenvolvedores, uso de testes automatizados

locais e aproveitamento de soluções já existentes. A partir dessas premissas, foram consideradas cinco estratégias diferentes, apresentadas na Tabela 6, como contribuições para o desenvolvimento de aplicativos móveis acessíveis.

Tabela 6. Possíveis estratégias para a real consideração do requisito acessibilidade.

Possíveis estratégias	
Orientação de testes	Indicação dos principais problemas de acessibilidade a serem verificados e sugestões de como abordar esses problemas, por papéis, com técnicas e ferramentas existentes.
Testes locais	Criar e incentivar mais testes locais, por meio de exemplos, atribuindo mais responsabilidade aos desenvolvedores.
Kit de testes	Implementar testes automatizados possíveis para cada papel, e disponibilizar suíte de testes *opensource*.
Pirâmide de testes	Distribuir soluções existentes na estrutura da pirâmide de testes
Extension Libs	Trabalhar com extensões dos *widgets* que ou já tratem possíveis problemas de acessibilidade, ou facilitem a realização de testes automatizados.

Com base no conhecimento técnico e científico adquirido nesta pesquisa, os autores deste trabalho sugerem que não seja solucionada uma das estratégias elencadas, mas sim, oferecer uma proposta abrangente. Essa proposta consiste em um *framework* conceitual que fornecerá aos pesquisadores e desenvolvedores um conjunto completo de conceitos, orientações, técnicas e ferramentas sobre acessibilidade. O *framework* deverá discutir sobre cada uma das estratégias apresentadas na Tabela 6, oferecendo soluções iniciais e apontando direções para futuras explorações. O conteúdo deverá ser oferecido de forma simples e prática, para incentivar o uso. Tal *framework* deverá ainda ser colaborativo e servir como um arcabouço teórico-prático para o desenvolvimento de aplicativos acessíveis para Android.

As subseções a seguir descrevem dois dos primeiros artefatos idealizados para o *framework*: definição de critérios de teste e a oferta de um kit de testes locais de acessibilidade.

4.1.1. Critérios de teste para recomendações de acessibilidade móvel

Como primeiro artefato do *framework* citado, foram reescritas as 25 recomendações de acessibilidade móvel (maRs) compiladas por Dias et al. [Dias et al. 2021], para pessoas com deficiência visual, no formato de critérios de sucesso [W3C 2022], ou seja, como declarações testáveis.

A reescrita das recomendações são apresentadas na Tabela 7, e podem ser utilizadas como um guia de requisitos de acessibilidade para que sejam trabalhados testes de acessibilidade no Android. Cada requisito foi escrito no formato: **alvo do teste [condição]** + *must|should* + **critério do teste**.

Esse conjunto de critérios reescritos tem como objetivo principal servir como referência prática para desenvolvedores e pesquisadores, além de orientar a criação de testes automatizados como os desenvolvidos no kit apresentado a seguir.

Tabela 7. Reescrita das maRs [Dias et al. 2021] como requisitos de acessibilidade.

Requisito de Acessibilidade
Elementos de texto devem utilizar uma taxa de contraste de pelo menos 4.5:1 entre a cor do texto e a cor do *background*.
Elementos de interação devem ter um tamanho mínimo de 48x48dp. Elementos de interação devem ter um espaçamento mínimo de 8dp da borda da tela e pelo menos 9dp de espaçamento entre eles.
Para cada controle de formulário deve haver um *TextView* com o atributo *labelFor* associado a ele ou deve ter o atributo *hint* fornecido.
Quando aplicável, os controles do formulário de dados de entrada devem fornecer dicas para conclusão. Quaisquer mensagens ou alterações na tela, como *pop-ups*, notificações e atualizações dinâmicas de conteúdo, devem ser anunciadas pelo leitor de tela.
Todas as ações do usuário executadas em segundo plano devem fornecer *feedback* de sucesso ou falha. Ações iniciadas a partir da interação do usuário, e que não podem ser desfeitas, devem solicitar confirmação antes de serem executadas. Todos os possíveis erros devem ser claramente informados ao usuário.
Todo conteúdo não textual deve ter uma descrição de texto alternativa.
Todas as telas do *app* devem ser redimensionáveis. Todas as *Views* personalizadas devem oferecer suporte à tecnologia assistiva.
Itens relacionados devem ser agrupados.
Todo elemento de interação deve permitir identificar o estado de acionabilidade, visualmente e por leitor de tela.
Todo o conteúdo que exceda as dimensões da tela (por exemplo, textos longos) deve ser apresentado usando representações alternativas apropriadas, como listas paginadas ou navegação hierárquica. Os parágrafos devem ter uma altura de interlinha de pelo menos 1,5 vezes o tamanho da fonte e o espaçamento dos parágrafos seguintes de pelo menos 2 vezes o tamanho da fonte. Campos de entrada de formulário devem indicar o formato de dados esperado, como datas e entradas numéricas. Toda a estrutura e layout do *app* devem permanecer consistentes em diferentes telas .
Todas as telas do *app* devem suportar navegação baseada em foco, destacando o componente focado. Todas as telas do *app* devem ter um título único e facilmente identificável na parte superior da tela. As telas internas do *app* devem fornecer um elemento de navegação para retornar à tela inicial.
Todas as telas do *app* devem manter um layout consistente em diferentes tamanhos e orientações de tela. Todos os elementos que dependem de informações sensoriais (como localização no mapa ou cores para representar informações) para que o conteúdo seja compreendido devem fornecer descrição textual associada. Campos de entrada de formulário devem garantir que o modo de teclado seja apropriado para o conteúdo do campo esperado.
A estrutura do formulário deve ser organizada com um campo por linha, evitando múltiplas colunas.

4.1.2. Kit de testes locais de acessibilidade

Com o objetivo de facilitar a verificação de requisitos de acessibilidade para pessoas com deficiência visual, por meio da condução de testes locais, foi desenvolvido um kit cujo foco se concentrou nos problemas de acessibilidade mais comuns e nos *widgets* mais usados: *TextView*, *ImageView*, *View*, *Button* e *ImageButton*. O *Automated accessibility tests kit for Android apps* (AATK), disponível no repositório GitHub[16], é uma biblioteca Android escrita em Java e disponibilizada publicamente para ajudar desenvolvedores na criação de testes de acessibilidade para projetos Android.

O kit contém inicialmente cinco testes para problemas de acessibilidade recorrentes que impactam negativamente a maneira como usuários com deficiência visual utilizam aplicativos nativos. Todos eles foram projetados para serem executados com o *Roboletric*, o que significa que podem ser executados dentro da JVM em segundos.

Esse conjunto inicial de testes é facilmente extensível, uma vez que cada teste implementa uma interface que facilita a execução por toda a hierarquia de *layout*, o que possibilita que novos testes podem ser criados a partir dessa interface.

O AATK facilita a verificação de requisitos de acessibilidade por meio de testes locais automatizados, que podem ser integrados à pipeline de CI/CD, promovendo maior

[16]

aderência à acessibilidade desde as fases iniciais do desenvolvimento.

4.2. Desdobramentos da pesquisa

Este trabalho faz parte das atividades de pesquisa de um grupo com foco no estudo da acessibilidade de soluções computacionais, em específico, nas soluções *mobile*. Tal grupo reúne profissionais da academia e da indústria, com diferentes olhares sobre o tema. É de interesse desse grupo fomentar as discussões sobre acessibilidade, mas ir além e oferecer recursos práticos e efetivos para que os profissionais da indústria se interessem, conheçam e implementem em suas soluções a acessibilidade.

Como parte das atividades do grupo, foram conduzidas avaliações do kit AATK com desenvolvedores de soluções *mobile*. Nessas avaliações, os participantes realizaram melhorias de acessibilidade em um aplicativo de exemplo, utilizando os testes oferecidos pelo kit como suporte. Grupos de controle executaram as mesmas tarefas com o auxílio do *Espresso* e do Scanner de Acessibilidade. Embora esses estudos forneçam insumos importantes, não constituem o foco deste artigo.

Além do kit de testes automatizados, também foram desenvolvidos outros artefatos no contexto do *framework*, como ferramentas para auxiliar *designers* na realização de testes de acessibilidade em componentes de *Design System*, recomendações de práticas e ferramentas voltadas a testadores, *codelabs* para treinamento e divulgação de testes de acessibilidade voltados a *designers*, desenvolvedores e testadores, além de um acervo com conteúdos derivados do trabalho, como publicações, guias, ferramentas, recomendações e exemplos. Esses materiais complementam as contribuições discutidas neste artigo, embora não sejam aqui detalhados.

Salienta-se aqui, que oferecer recursos de acessibilidade em soluções computacionais é uma condição prevista em lei no Brasil e, para além da empatia com o tema, é preciso que profissionais que produzem tecnologias levem a implementação do requisito mais a sério. Os estudos e entrevistas conduzidas no escopo deste trabalho, somados à experiência dos autores, permitem afirmar que esse requisito não tem sido implementado, na maioria da vezes, pela falta de conhecimento dos profissionais da área. É preciso um esforço conjunto da indústria e da academia para, não só discutir sobre o tema e propor soluções pontuais, mas realizar ações práticas como incentivar disciplinas específicas de acessibilidade nas grades dos cursos da Computação e áreas afins, ou reforçar que esse conteúdo seja ministrado nas disciplinas de IHC (neste sentido, seria preciso ações práticas de sensibilização e/ou formação de professores sobre o tema); além de propor um ferramental que apoie os profissionais envolvidos, oferecendo formas mais práticas de aplicação das técnicas.

Este grupo de pesquisadores acredita, por fim, que a nova norma ABNT NBR 17225:2025 - Acessibilidade em conteúdo e aplicações web - Requisitos, de março de 2025[17], possa corroborar na disseminação e fiscalização da implementação desse requisito nas soluções computacionais.

5. Cuidados Éticos

Este trabalho foi aprovado pelo Comitê de Ética em Pesquisa em Seres Humanos (CEP) da Universidade Federal de São Carlos (UFSCar) sob o número CAAE

[17]Disponível em: `https://www.abntcolecao.com.br/mpf/norma.aspx?ID=567818`

71178423.7.0000.5504. Os participantes assinaram os termo de consentimento e forma explicados sobre os objetivos da pesquisa, que eles eram voluntários e não envolvia pagamento, que seus dados seriam anonimizados e que eles poderiam interromper a participação a qualquer momento, sem prejuízo para os mesmos Os autores reiteram o compromisso desta pesquisa com os padrões de integridade científica e respeito aos direitos e bem-estar dos participantes envolvidos.

6. Agradecimentos

Agradecemos a todos os voluntários que participaram das atividades envolvidas neste trabalho.

Nota: Este artigo não fez uso de ferramentas de IA para produção de conteúdo e nem de correção ortográfica, mas fez uso do ChatGPT e Gemini para descrição das imagens.

Referências

Acosta-Vargas, P., Salvador-Ullauri, L., Jadán-Guerrero, J., Guevara, C., Sanchez-Gordon, S., Calle-Jimenez, T., Lara-Alvarez, P., Medina, A., e Nunes, I. L. (2020). Accessibility assessment in mobile applications for android. *Advances in Intelligent Systems and Computing*, 959:279–288.

Alshayban, A., Ahmed, I., e Malek, S. (2020). 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. ACM.

Aniche, M. (2014). *Real World Test-Driven Development*. Casa do Código, São Paulo.

Apple (2022). Accessibility - apple developer. Disponível em https://developer.apple.com/accessibility/. Acesso em 01 de maio 2025.

Ballantyne, M., Jha, A., Jacobsen, A., Hawker, J. S., e El-Glaly, Y. N. (2018). Study of accessibility guidelines of mobile applications. In *Proc of the 17th Intl Conf on Mobile and Ubiquitous Multimedia*, MUM 2018, page 305–315, New York, NY, USA. ACM.

Baranauskas, M. C. C., de Souza, C. S., e Pereira, R. (2012). Grandihc-br: prospecção de grandes desafios de pesquisa em interação humano-computador no brasil. In *Companion Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems*, pages 63–64.

BBC (2022). Accessibility - bbc. Disponível em https://www.bbc.co.uk/accessibility/. Acesso em 10 de julho 2023.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). The agile manifesto. Disponível em http://www.agilemanifesto.org. Acesso em 10 de julho 2023.

Bi, T., Xia, X., Lo, D., Grundy, J., Zimmermann, T., e Ford, D. (2022). Accessibility in software practice: A practitioner's perspective. *Trans. on Software Engineering and Methodology*, 31:1–26.

BRASIL (2015). Lei nº 13.146, de 6 de julho de 2015 (lei brasileira de inclusão da pessoa com deficiência / estatuto da pessoa com deficiência).

Disponível em http://www2.camara.leg.br/legin/fed/lei/2015/lei-13146-6-julho-2015-781174-normaatualizada-pl.pdf. Acesso em 01 de maio 2025.

Carvalho, L. P., Peruzza, B. P. M., Santos, F., Ferreira, L. P., e Freire, A. P. (2016). Accessible smart cities?: Inspecting the accessibility of brazilian municipalities' mobile applications. In *Proc of the 15th Brazilian Symposium on Human Factors in Computing Systems Human Factors in Computing Systems*, pages 17:1–17:10, New York, NY, USA. ACM.

Carvalho, M. C. N., Dias, F. S., Reis, A. G. S., e Freire, A. P. (2018). Accessibility and usability problems encountered on websites and applications in mobile devices by blind and normal-vision users. In *Proc of the 33rd Annual Symposium on Applied Computing*, pages 2022–2029, New York, NY, USA. ACM.

Crispin, L. e Gregory, J. (2009). *Agile testing: A practical guide for testers and agile teams*. Pearson Education, Boston, MA.

Cruz, L., Abreu, R., e Lo, D. (2019). To the attention of mobile software developers: guess what, test your app! *Empirical Software Engineering 2019 24:4*, 24:2438–2468.

Curcio, K., Santana, R., Reinehr, S., e Malucelli, A. (2019). Usability in agile software development: A tertiary study. *Computer Standards & Interfaces*, 64:61–77.

Damaceno, R. J. P., Braga, J. C., e Mena-Chalco, J. P. (2018). Mobile device accessibility for the visually impaired: problems mapping and recommendations. *Universal Access in the Information Society*, 17:421–435.

de Oliveira, A. F. B. A. e Filgueiras, L. V. L. (2018). Developer assistance tools for creating native mobile applications accessible to visually impaired people: A systematic review. In *Intl Conf. Proc Series*, IHC 2018, pages 16:1—-16:9, New York, NY, USA. ACM.

de Oliveira, A. F. B. A. e Filgueiras, L. V. L. (2019). Accessibilint: A tool for early accessibility verification for android native applications. Association for Computing Machinery.

de Oliveira, L. C., Amaral, M. A. a., Bim, S. A., Valença, G., Almeida, L. D. A., Salgado, L. C. d. C., Gasparini, I., e da Silva, C. B. R. (2024). Grandihc-br 2025-2035 - gc3: Plurality and decoloniality in hci. In *Proceedings of the XXIII Brazilian Symposium on Human Factors in Computing Systems*, IHC '24, New York, NY, USA. Association for Computing Machinery.

Delamaro, M. E., Maldonado, J. C., e Jino, M. (2016). *Introdução ao teste de software*. Elsevier, Rio de Janeiro.

Developers, S. (2022). Accessibility introduction | samsung developers. Disponível em https://developer.samsung.com/one-ui/accessibility/intro.html. Acesso em 01 de maio 2025.

Dias, F., Duarte, L., e Fortes, R. (2021). Accessmdd: An mdd approach for generating accessible mobile applications. *Proc of the 39th ACM Intl Conf on the Design of Communication - SIGDOC 2021*, 11:85–95.

Dias, F. S. (2021). *AccessMDD: uma abordagem MDD para geração de aplicativos móveis acessíveis*. PhD thesis, Universidade de São Paulo.

Eckhardt, J., Vogelsang, A., e Fernández, D. M. (2016). Are non-functional requirements really non-functional? an investigation of non-functional requirements in practice. *Proceedings - International Conference on Software Engineering*, 14-22-May-2016:832–842.

Eler, M. M., Rojas, J. M., Ge, Y., e Fraser, G. (2018). Automated accessibility testing of mobile apps. In *2018 IEEE 11th Intl Conf on Software Testing, Verification and Validation (ICST)*, pages 116–126, Västerås, Sweden. IEEE Inc.

Gomes, B., Rios, J., e Rodrigues, K. R. (2020). Challenges for the implementation of accessible web and mobile systems. In *Software Ecosystems, Sustainability and Human Values in the Social Web: 8th Workshop of Human-Computer Interaction Aspects to the Social Web, WAIHCWS 2017, Joinville, Brazil, October 23, 2017 and 9th Workshop, WAIHCWS 2018, Belém, Brazil, October 22, 2018, Revised Selected Papers 8*, pages 138–158. Springer.

Gomes, F. T., de Lima Salgado, A., Duarte, L. M. C., Santos, F. S., e Fortes, R. P. M. (2018). Um simulador visual de leitor de telas para auxílio à interpretação de questões de acessibilidade por avaliadores videntes. *Revista de Sistemas e Computação-RSC*, 8(1):114–134.

Google (2022). Accessibility test framework for android. Disponível em https://github.com/google/Accessibility-Test-Framework-for-Android. Acesso em 10 de julho 2023.

Google (2023a). Developer guides. Disponível em https://developer.android.com/guide/. Acesso em 01 de maio 2025.

Google (2023b). Roboletric. Disponível em https://github.com/robolectric/robolectric. Acesso em 10 de julho 2023.

Hess, S., Kiefer, F., Carbon, R., e Maier, A. (2013). nconcappt - a method for the conception of mobile business applications. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, 110 LNICST:1–20. cited By 7.

ISO (2008). Ergonomics of human-system interaction — part 171: Guidance on software accessibility. Technical Report ISO 9241-171:2008, International Organization for Standardization.

Kitchenham, B. A., Budgen, D., e Brereton, O. P. (2010). The value of mapping studies – a participant-observer case study. BCS Learning and Development.

Kochhar, P. S., Thung, F., Nagappan, N., Zimmermann, T., e Lo, D. (2015). Understanding the test automation culture of app developers. In *8th Intl Conf on Software Testing, Verification and Validation (ICST)*, pages 1–10, Graz, Austria. IEEE.

Leite, M. V. R., Scatalon, L. P., Freire, A. P., e Eler, M. M. (2021). Accessibility in the mobile development industry in brazil: Awareness, knowledge, adoption, motivations and barriers. *Journal of Systems and Software*, 177:110942.

Losada, B., Fernández-Castro, I., López-Gil, J.-M., e Urretavizcaya, M. (2013). Applying usability engineering in intermod agile development methodology. a case study in a

mobile application. *JUCS - Journal of Universal Computer Science*, 19(8):1046–1065. Place: Austria Publisher: Verlag der Technischen Universität Graz.

Mankoff, J., Hofmann, M., Chen, X. A., Hudson, S. E., Hurst, A., e Kim, J. (2019). Consumer-grade fabrication and its potential to revolutionize accessibility. *Communications of the ACM*, 62:64–75.

Nonaka, I. e Takeuchi, H. (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, New York.

Oliveira, R. e França, C. (2019). Agile practices and motivation: A quantitative study with brazilian software developers. In *Proc of the Evaluation and Assessment on Software Engineering*, EASE '19, page 365–368, New York, NY, USA. ACM.

Paiva, D. M. B., Freire, A. P., e Fortes, R. P. M. (2021). Accessibility and software engineering processes: A systematic literature review. *Journal of Systems and Software*, 171:1–17.

Pellegrini, F., Anjos, M., Florentin, F., Ribeiro, B., Correia, W., e Quintino, J. (2020). How to prioritize accessibility in agile projects. In Ahram, T. e Falcão, C., editors, *Advances in Usability and User Experience*, pages 271–280, Cham. Springer.

Petrie, H. e Bevan, N. (2009). The Evaluation of Accessibility, Usability, and User Experience. In Stephanidis, C., editor, *The Universal Access Handbook*, volume 20091047, pages 1–16. CRC Press.

Pressman, R. S. (2011). *Engenharia de Software: Uma Abordagem Profissional*. AMGH, Porto Alegre, 7 edition.

Rieger, C., Lucrédio, D., Fortes, R. P. M., Kuchen, H., Dias, F., e Duarte, L. (2020). A model-driven approach to cross-platform development of accessible business apps. In *Proc of the 35th Annual ACM Symposium on Applied Computing*, pages 984–993, New York, NY, USA. ACM.

Salman, F. A. e Deraman, A. (2022). A model for incorporating suitable methods of usability evaluation into agile software development. *Bulletin of Electrical Engineering and Informatics*, 11:3433–3440.

Samsung Instituto de Desenvolvimento para Informática (SIDI) (2017). Guia para o desenvolvimento de aplicações móveis acessíveis. Disponível em https://www.sidi.org.br/guiadeacessibilidade/index.html. Acesso em 20 de julho 2023.

Siebra, C., Gouveia, T. B., Macedo, J., Silva, F. Q. B. D., Santos, A. L. M., Correia, W., Penha, M., Florentin, F., e Anjos, M. (2017). 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. ACM.

Swallow, D., Petrie, H., e Power, C. (2016). Understanding and supporting web developers: Design and evaluation of a web accessibility information resource (webair). *Studies in Health Technology and Informatics*, 229:482–491.

Thatcher, J., Bohman, P., Burks, M., Henry, S. L., Regan, B., Swierenga, S., Urban, M. D., e Waddell, C. D. (2002). *Constructing Accessible Web Sites*. Apress, Berkeley, CA.

Turine, M. A. S. e Masiero, P. C. (1996). Especificacao de requisitos: uma introducao. Disponível em https://repositorio.usp.br/item/000906321. Acesso em 25 de outubro 2024.

Vendome, C., Solano, D., Liñán, S., e Linares-Vásquez, M. (2019). Can everyone use my app? an empirical study on accessibility in android apps. In *2019 IEEE Intl Conf on Software Maintenance and Evolution (ICSME)*, pages 41–52, Cleveland, OH, USA. IEEE.

Vontell, A. R. (2019). *Bility : Automated Accessibility Testing for Mobile Applications by*. PhD thesis, Massachusetts Institute of Technology.

W3C (2022). World wide web consortium. Disponível em https://www.w3.org/. Acesso em 01 de maio 2025.

Winkelmann, H., Troost, L., e Kuchen, H. (2022). Constraint-logic object-oriented programming for test case generation. In *Proc of the 37th ACM/SIGAPP Symposium on Applied Computing*, page 1499–1508, New York, NY, USA. Association for Computing Machinery.

Wolkerstorfer, P., Tscheligi, M., Sefelin, R., Milchrahm, H., Hussain, Z., Lechner, M., e Shahzad, S. (2008). Probing an agile usability process. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '08, page 2151–2158, New York, NY, USA. Association for Computing Machinery. Acesso em 04 de julho 2023.

Yan, S. e Ramachandran, P. G. (2019). The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing*, 12(1).

Yin, R. K. (2016). *Pesquisa qualitativa do início ao fim*. Penso Editora.