Universidade do Estado da Bahia

Departamento de Ciências Exatas e da Terra

Colegiado de Sistemas de Informação

DANILO SIMAS GONÇALVES

Um Guia Baseado em Heurísticas de Usabilidade Específicas para o Desenvolvimento de Aplicações para Dispositivos Móveis

Salvador

2018

DANILO SIMAS GONÇALVES

Um Guia Baseado em Heurísticas de Usabilidade Específicas para o Desenvolvimento de Aplicações para Dispositivos Móveis

Trabalho de Conclusão de Curso apresentado à banca do curso de Bacharelado em Sistemas de Informação da Universidade do Estado da Bahia, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.
Orientador: Prof. Dr. Eduardo Manuel de Freitas Jorge

Salvador
2018

Gonçalves, Danilo Simas

Um Guia Baseado em Heurísticas de Usabilidade Específicas para o Desenvolvimento de Aplicações para Dispositivos Móveis: / Danilo Simas Gonçalves. – Salvador, 2018.

52 fls. : il.

Orientador: Prof. Dr. Eduardo Manuel de Freitas Jorge

Trabalho de Conclusão de Curso (Bacharelado) – Universidade do Estado da Bahia. Departamento de Ciências Exatas e da Terra. Colegiado de Sistemas de Informação, 2018.

1. Usabilidade. 2. Heurísticas de Usabilidade. 3. Smartphones. 4. Guia. 5. Dispositivos Móveis. 6. Componentes. I. Universidade do Estado da Bahia. Departamento de Ciências Exatas e da Terra.

CDD: 025.06

DANILO SIMAS GONÇALVES

Um Guia Baseado em Heurísticas de Usabilidade Específicas para o Desenvolvimento de Aplicações para Dispositivos Móveis

Trabalho de Conclusão de Curso apresentado à banca do curso de Bacharelado em Sistemas de Informação da Universidade do Estado da Bahia, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.

Trabalho aprovado. Salvador, 27 de junho de 2018:

Prof. Dr. Eduardo Manuel de Freitas Jorge
Professor Orientador

Prof. Dr. Ernesto de Souza Massa Neto
Professor Convidado

Prof. MSc. Alexandre Rafael Lenz
Professor Convidado

Salvador
2018

Resumo

Com a popularização dos *smartphones* também cresce o número de aplicativos disponíveis no mercado para esta plataforma, apesar desse contínuo crescimento muitos aplicativos ainda estão falhando em manter os usuários interessados e envolvidos por muito tempo. Parte disso se dá pelo fato dos *smartphones* possuírem características muito distintas dos tradicionais *desktops*, o que implica em uma série de desafios para o design e desenvolvimento de aplicativos com boa usabilidade. Para vencer esses desafios e desenvolver aplicações com boa usabilidade recorre-se às heurísticas de usabilidade como guia na construção e avaliação dessas aplicações. No entanto, atualmente, as heurísticas disponíveis e já consolidadas são gerais e não tratam de todos os aspectos característicos dessa plataforma, além disso, há uma carência de componentes que se predispõem a atender heurísticas de usabilidade. Nesse contexto, este trabalho propõe a elaboração de um guia de usabilidade, independente do sistema operacional, que incorpore um conjunto de heurísticas adequadas para os aspectos característicos dos *smartphones*. Além do guia foram desenvolvidos alguns componentes que atendem às especificações do guia proposto visando auxiliar os desenvolvedores, numa perspectiva de reúso, na construção de suas aplicações.

Palavras-chave: Usabilidade, Heurísticas de Usabilidade, Smartphones, Guia, Dispositivos Móveis, Componentes.

Abstract

With the popularization of smartphones also grows the number of applications available in the market for this platform, despite this continued growth many applications are still failing to keep users interested and involved for a long time. Part of this is due to the fact that smartphones have very different characteristics from traditional desktops, which implies in a series of challenges for the design and development of applications with good usability. To overcome these challenges and to develop applications with good usability we refer to usability heuristics as a guide in the construction and evaluation of these applications. However, the available and already consolidated heuristics are general and do not deal with all aspects of the platform, in addition, there is a lack of components that are predisposed to incorporate usability heuristics. In this context, this paper proposes the development of a usability guide, independent of the operating system, that incorporates a set of heuristics suitable for the characteristic aspects of smartphones. In addition to the guide, some components have been developed that meet the specifications of the proposed guide in order to help developers, in a reuse perspective, in the construction of their applications.

Keywords: Usability, Usability Heuristics, Smartphones, Guide, Mobile Devices, Components.

Lista de ilustrações

Figura 1 – Estrutura do guia 24
Figura 2 – Exemplo de uma aplicação que trata o estado vazio 26
Figura 3 – Tamanho da área de toque 29
Figura 4 – Telas dos aplicativos que não atenderam a recomendação R5 33
Figura 5 – Telas dos aplicativos que não atenderam a recomendação R6 34
Figura 6 – Telas dos aplicativos que não atenderam a recomendação R7 35
Figura 7 – Telas dos aplicativos quando a lista de horários está vazia 37
Figura 8 – Telas dos aplicativos com um horário sendo exibido na lista 37

Lista de tabelas

Tabela 1 – Características essenciais de um componente de software 19
Tabela 2 – Benefícios do reúso de software 20
Tabela 3 – Atendimento dos aplicativos às recomendações 36

Sumário

1 INTRODUÇÃO 9

2 INTERFACE HUMANO-COMPUTADOR 11
2.1 Usabilidade 11
2.2 Heurísticas de Usabilidade 13
2.3 Aspectos dos Smartphones 15

3 ENGENHARIA DE SOFTWARE 17
3.1 Reúso 17
3.2 Componentes de Software 17
3.3 Padrões de Projeto 18

4 GUIA DE USABILIDADE 21
4.1 Trabalhos Correlatos 21
4.2 Percurso Metodológico 23
4.3 Elaboração do Guia 23
4.4 Desenvolvimento dos Componentes 25
4.4.1 Componente LiloRecyclerView 26
4.4.2 Componente LiloImageButton 28
4.5 Experimentos 31
4.6 Análise dos Resultados 32

5 CONSIDERAÇÕES FINAIS 38

REFERÊNCIAS 39

APÊNDICES 41

APÊNDICE A – COMPONENTE LILORECYCLERVIEW 42

APÊNDICE B – COMPONENTE LILOIMAGEBUTTON 44

ANEXOS 47

ANEXO A – HEURÍSTICAS ESPECÍFICAS PARA DISPOSITIVOS MÓVEIS SELECIONADAS 48

1 Introdução

Os *smartphones*, diferente dos tradicionais telefones celulares, são dispositivos móveis com um grande poder computacional capazes de mais do que realizar chamadas e enviar mensagens. Por intermédio de seus aplicativos é possível realizar tarefas que antes só eram possíveis através de computadores *desktops*, como: fazer compras, pagar contas, acessar e-mail, assistir filmes, pesquisar, jogar e entre outras coisas. Os aplicativos móveis mudaram a forma como percebemos e interagimos com o mundo, auxiliando e facilitando nas tarefas e nas relações do dia-a-dia.

Com a popularização desses dispositivos também cresce o número de aplicativos disponíveis no mercado. De acordo com a Statista (2016b) só a *Google Play* e a *Apple Store*, líderes do mercado de aplicativos, totalizam cerca de 4,2 milhões de aplicativos em suas lojas e , segundo Emarketer (2014), estima-se que em 2017 aproximadamente 200 bilhões de downloads de aplicativos foram feitos, cerca de duas vezes e meia o número de 2013. Apesar desse contínuo crescimento, a maioria dos aplicativos estão falhando em manter os usuários interessados e envolvidos por muito tempo, como revela uma análise conduzida pela *Localytics*, plataforma que fornece análises do comportamento de usuários móveis. Segundo a análise, cerca de 23% dos usuários de aplicativos móveis no mundo desistiram de um aplicativo depois de usá-lo apenas uma vez (EMARKETER, 2016).

Já uma pesquisa realizada pela Google e Ipsos (2015) relatou que apenas 26% dos aplicativos instalados em um smartphone são usados diariamente. Dos 6 atributos associados com os aplicativos usados frequentemente, observa-se que 4 estão diretamente relacionados a uma boa usabilidade e experiência do usuário. Desse modo, a atenção à usabilidade representa um cuidado importante na construção de aplicativos atraentes, como confirma Gove (2016):

> Mais do que nunca, as pessoas estão se engajando com seus telefones em momentos cruciais e por períodos mais curtos de tempo. Suas experiências precisam ser eficientes e agradáveis. Além disso, um aplicativo bem projetado que fornece utilidade tem o poder de se destacar dos demais. (GOVE, 2016)

O advento dos smartphones trouxe um novo modelo de interação e com ele alguns aspectos característicos que trazem desafios no design de interfaces para esses dispositivos, como: as limitações físicas e tecnológicas, as particularidades e objetivos dos usuários, além de um contexto extremamente dinâmico. Para vencer esses desafios e desenvolver aplicações com boa usabilidade recorre-se às heurísticas de usabilidade como guia na construção e avaliação dessas aplicações, dentre as quais destaca-se as dez heurísticas de Nielsen (1995). No entanto, atualmente, as heurísticas disponíveis já consolidadas são

gerais e não tratam de todos os aspectos característicos desses dispositivos, como relata a pesquisa de Salazar et al. (2015). Além disso, há uma carência de componentes que se predispõem a atender heurísticas de usabilidade. Como consequência, o desenvolvimento de aplicações com bons critérios de usabilidade ainda demanda muito esforço e tempo dos desenvolvedores.

Nesse mercado competitivo de aplicativos móveis o desenvolvimento de aplicações com uma boa usabilidade e que proporcionem uma boa experiência aos usuários apresenta-se como um dos fatores cruciais para se destacar e reter usuários. Em vista disso, existe um esforço das principais empresas de sistemas operacionais para plataforma móvel, Apple e Google, em fornecer recomendações que reúnem princípios de design e usabilidade para orientar designers e desenvolvedores na criação de aplicativos capazes de prover uma melhor experiência aos seus usuários. Apesar desta iniciativa contribuir até certo ponto para o desenvolvimento de interfaces mais usáveis, esses guias trazem como foco a padronização da apresentação, interação e utilização dos seus componentes.

Diante desse cenário, o objetivo deste trabalho consiste na elaboração de um guia de usabilidade, independente do sistema operacional, que incorpore um conjunto de heurísticas adequadas para os aspectos característicos dos *smartphones*. Para isso, além do guia foram desenvolvidos alguns componentes que atendem às especificações do guia proposto, com o intuito de auxiliar os desenvolvedores, numa perspectiva de reúso, no desenvolvimento dessas aplicações.

Assim, espera-se que esse trabalho apoie os desenvolvedores no processo de construção de aplicações com uma usabilidade adequada e na redução de esforço de teste e correção de problemas de interação dos usuários com as aplicações, e diante da relevância do tema, contribua para discussões sobre heurísticas de usabilidade específicas e de componentes que as incorporam.

O presente trabalho está disposto em seis capítulos. O Capítulo 2 aborda sobre os temas relacionados a área de IHC (Interface Humano-Computador): características dos *smartphones*, usabilidade, heurísticas de usabilidade, e trabalhos correlatos. O Capítulo 3 trata sobre os temas ligados a Engenharia de Software: reúso, o desenvolvimento de componentes de software e técnicas utilizadas no desenvolvimento do trabalho. O Capítulo 4 traz o percurso metodológico e contempla todo a etapa de projeto. E por último no Capítulo 5 encontram-se as considerações finais.

2 Interface Humano-Computador

Este capítulo apresenta a definição de usabilidade no contexto de sistemas interativos, aborda sobre as heurísticas de usabilidade e como elas são utilizadas na avaliação e concepção desses sistemas, e discorre sobre os aspectos característicos dos *smartphones* e como eles impactam na usabilidade e no desenvolvimento de aplicações para esses dispositivos.

2.1 Usabilidade

A usabilidade pode ser definida como um atributo de qualidade que avalia o quão fáceis as interfaces de usuário são de usar (NIELSEN, 2012). Para Preece, Rogers e Sharp (2005, p. 35-36), no contexto de sistemas interativos, a usabilidade é dividida em seis metas que, quando alcançadas, contribuem para a concepção de sistemas mais usáveis. São elas:

1. **Eficácia:** o quanto o sistema atende aos resultados esperados - faz o que se propõe a fazer;
2. **Eficiência:** o quanto o sistema auxilia os usuários na realização de suas tarefas - ajuda a realizar as tarefas com menos esforço e de uma forma mais fácil.
3. **Segurança:** o quanto o sistema protege o usuário de realizar ações acidentais e de provocar situações indesejadas - o quanto o sistema ajuda na prevenção e na remediação de erros indesejáveis;
4. **Utilidade:** o quanto o sistema fornece de funcionalidades para que os usuários sejam capazes de realizar as tarefas.
5. **Capacidade de aprendizagem:** quão fácil é de aprender a usar o sistema.
6. **Capacidade de memorização:** quão fácil é de lembrar como utilizar o sistema, uma vez que já foi aprendido.

Outra forma de conceitualizar usabilidade é através de princípios. Os princípios implicam a orientação geral pretendida para informar o design e a avaliação de um sistema (PREECE; ROGERS; SHARP, 2005, p. 42). Em outras palavras, os princípios são responsáveis por nortear designers e desenvolvedores na construção e na avaliação de interfaces com uma melhor usabilidade. De acordo com Preece, Rogers e Sharp (2005, p. 42) vários princípios já foram desenvolvidos e os mais conhecidos referem-se a como determinar o que os usuários devem ver e fazer enquanto interagem com um produto. Um

dos conjunto de princípios mais conhecidos é o proposto por Norman (2002) em seu livro "*The design of everyday things*", são eles:

1. **Visibilidade:** é o princípio básico de que quanto mais visível for um elemento, mais provável será que os usuários saibam sobre eles e como usá-los. Igualmente importante é o oposto: quando algo está fora de vista, é difícil saber e usar.

2. ***Feedback:*** é o princípio de tornar claro para o usuário que ação foi tomada e o que foi realizado. Muitas formas de *feedback* existem no design de interação, incluindo visual, tátil, de áudio e muito mais. A chave é projetar a experiência para nunca deixar o usuário adivinhando a ação que ele tomou e a consequência de fazê-lo.

3. **Restrições:** é o princípio de limitar a gama de possibilidades de interação para simplificar a interface e guiar o usuário para a próxima ação apropriada. Este é um caso em que as restrições são esclarecedoras, pois deixam claro o que pode ser feito. Possibilidades ilimitadas muitas vezes deixam o usuário confuso.

4. **Mapeamento:** é o princípio de ter uma relação clara entre os controles e o efeito que eles têm no mundo. O mapeamento deve ser o mais natural possível.

5. **Consistência:** é o princípio que refere-se a ter operações semelhantes e elementos semelhantes para alcançar tarefas semelhantes. Ao alavancar elementos consistentes ao longo de toda a sua experiência, você torna sua experiência muito mais fácil de usar.

6. ***Affordance:*** é o princípio que refere-se a um atributo de um objeto que permite que as pessoas saibam como usá-lo. Essencialmente, *affordance* significa dar uma pista.

Segundo Preece, Rogers e Sharp (2005, p. 41-48) enquanto os princípios de design tratam de abstrações generalizáveis destinadas a orientar os designers a pensar sobre aspectos diferentes de seus designs, sugerindo o que utilizar e o que evitar na construção de uma interface, os princípios de usabilidade tendem a ser mais prescritivos e são utilizados sobretudo como base para a avaliação de protótipos e sistemas existentes. Para ilustrar essa diferença, podemos considerar um princípio de design como: "O *feedback* é um retorno de informação que ajuda o usuário a entender o resultado de suas ações". Ao passo que um princípio de usabilidade como sendo: "Mantenha o usuário sempre informado sobre o que está acontecendo, fornecendo o *feedback* adequado, dentro de um tempo razoável".

Apesar de seguir os princípios ser um passo importante na construção de interfaces com melhor usabilidade, esta não é suficiente para assegurar que as interfaces desenvolvidas apresentam de fato uma experiência agradável e satisfatória aos usuários, para isso, se faz

necessária a avaliação. As avaliações de usabilidade são técnicas responsáveis por auxiliar na identificação de problemas de usabilidade. De acordo com Preece, Rogers e Sharp (2005, p. 338) a avaliação é definida como:

> `[...]` o processo sistemático de coleta de dados responsável por nos informar o modo como um determinado usuário ou grupo de usuários deve utilizar um produto para uma determinada tarefa em um certo tipo de ambiente. (Preece et al., 2015, p. 338)

Uma possível forma de classificar as técnicas de avaliação de usabilidade é quanto à participação de usuários: enquanto as técnicas com participação de usuários são aquelas que envolvem a observação da interação do usuário com o sistema, as técnicas sem participação de usuários são aquelas onde especialistas em usabilidade assumem o papel de usuários. As avaliações que em geral não desfrutam da participação de usuários e sim da opinião de especialistas são comumente conhecidas como métodos de inspeção.

Os métodos de inspeção são apropriados em circunstâncias em que os usuários não estão facilmente acessíveis, ou envolvê-los é muito caro ou requer muito tempo. Esse tipo de método tem como base a avaliação preditiva, onde um conjunto de especialistas inspecionam a interface homem-computador e preveem problemas que os usuários iriam se deparar ao interagir com ela (PREECE; ROGERS; SHARP, 2005, p. 429-430). São alguns desses métodos a avaliação heurística e os *walkthroughs* (passo a passo).

2.2 Heurísticas de Usabilidade

Dentre os métodos de inspeção de usabilidade destaca-se a avaliação heurística, proposta por Nielsen e Molich (1990). Essa avaliação constitui-se em um método onde especialistas, orientados por um conjunto de princípios de usabilidade conhecidos como heurísticas, avaliam se os elementos da interface com o usuário estão de acordo com estes princípios. Essa técnica é amplamente conhecida por possibilitar a identificação dos principais problemas de usabilidade em uma interface, por ela ser um método rápido, fácil de aplicar e de baixo custo, quando comparado a outros métodos (PREECE; ROGERS; SHARP, 2005, p. 430).

O conjunto de heurísticas de usabilidade mais popular é o conjunto de dez heurísticas propostas por Nielsen (1995). Segundo Nielsen, apesar de suas heurísticas serem originadas dos sistemas desktop, elas são básicas e universais de maneira a se aplicar a diferentes sistemas (PREECE; ROGERS; SHARP, 2005, p. 447). São elas:

1. **Visibilidade do status do sistema:** O sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de *feedback* apropriado num prazo razoável.

2. **Correspondência entre o sistema e o mundo real:** O sistema deve falar a linguagem dos usuários, com palavras, frases e conceitos familiares para o usuário, ao invés de termos orientados ao sistema. Siga as convenções do mundo real, fazendo com que as informações apareçam em uma ordem natural e lógica.

3. **Controle e liberdade do usuário:** Os usuários costumam escolher funções do sistema por engano e precisam de uma "saída de emergência" claramente marcada para deixar o estado indesejado sem ter que passar por um extenso diálogo. Dê suporte ao desfazer e refazer.

4. **Consistência e padrões:** Os usuários não devem ter que se perguntar se diferentes palavras, situações ou ações significam a mesma coisa. Siga as convenções da plataforma.

5. **Prevenção de erro:** Ainda melhor do que boas mensagens de erro é um design cuidadoso que impede que um problema ocorra em primeiro lugar. Elimine as condições passíveis de erros ou verifique que elas apresentem aos usuários uma opção de confirmação antes de se comprometer com a ação.

6. **Reconhecimento ao invés de recordação:** Minimizar a carga de memória do usuário, fazendo objetos, ações e opções visíveis. O usuário não deve ter que se lembrar de informações a partir de uma parte do diálogo para outro. Instruções para a utilização do sistema deve ser visível ou facilmente recuperáveis sempre que adequado.

7. **Flexibilidade e eficiência de uso:** Aceleradores - invisíveis pelo usuário iniciante - muitas vezes podem acelerar a interação para o usuário experiente tal que o sistema pode atender a ambos os usuários inexperientes e experientes. Permita que usuários se adaptem às ações frequentes.

8. **Estética e design minimalista:** Diálogos não devem conter informações irrelevantes ou raramente necessárias. Cada unidade extra de informação em um diálogo compete com as unidades de informação relevantes e diminui sua visibilidade relativa.

9. **Ajudar os usuários a reconhecer, diagnosticar e recuperar de erros:** Mensagens de erro devem ser expressas em linguagem simples (sem códigos), indicar com precisão o problema e construtivamente sugerir uma solução.

10. **Ajuda e documentação:** Mesmo que seja melhor que o sistema possa ser usado sem documentação, pode ser necessário fornecer ajuda e documentação. Qualquer informação deve ser fácil de pesquisar, focada na tarefa do usuário, listar passos concretos a realizar, e não ser muito extensa.

De acordo com Preece, Rogers e Sharp (2005, p. 431) esse conjunto original de heurísticas proposta por Nielsen são muito gerais para avaliar novos produtos e as características específicas que não são abordadas por essas heurísticas genéricas podem ter um grande impacto na experiência dos usuários. Assim, eles acreditam que existe uma forte necessidade de heurísticas específicas que sejam moldadas às características de novos produtos.

2.3 Aspectos dos Smartphones

Diferente dos tradicionais *desktops*, os *smartphones* apresentam um conjunto de aspectos que fazem com que projetar interfaces para esses dispositivos possa ser considerado uma tarefa desafiadora. Um artigo publicado pelo Nielsen Norman Group (BUDIU, 2015), renomado grupo de pesquisa na área de usabilidade, reúne alguns aspectos de usabilidade dos *smartphones* e discute como eles influenciam na experiência dos usuários e no design de interfaces para esses dispositivos. Em seguida encontram-se dispostos os principais pontos discutidos neste artigo.

Ainda que os *smartphones* tenham aumentado de tamanho nos últimos tempos, para permanecerem convenientes e portáveis eles demandam uma tela reduzida, por conta disso, o tamanho da tela ainda representa uma séria limitação para esses dispositivos. Como consequência dessa limitação as telas tem capacidade de acomodar muito menos conteúdo do que os aparelhos *desktops* e *laptops*, por exemplo. Com menos conteúdo sendo apresentado na tela é exigido do usuário mais interações para ter acesso à mesma quantidade de informação, ou seja, é preciso de mais telas ou mais toques para ter acesso ao mesmo conteúdo que teria em um aparelho *desktop*, por exemplo, e uma maior utilização da memória a curto prazo do usuário para se referir às informações que não estão visíveis na tela.

Por serem projetados para serem facilmente portáveis, a ponto de caber em um bolso da roupa, os *smartphones* permitem que o seu uso se dê em contextos diversos e dinâmicos, o que torna sua interação mais suscetível a interrupções. Como resultado disso, a atenção do usuário é geralmente fragmentada e as sessões de interação com o aparelho são curtas. Dadas essas circunstâncias, certos cuidados fazem-se necessários na hora de projetar interfaces para esses dispositivos, como a necessidade de armazenar constantemente o estado atual do usuário no sistema e a necessidade de construir interfaces mais objetivas e focadas em uma tarefa clara.

Devido a sua tela ser sensível ao toque, a principal forma de interação com os *smartphones* se dá através do dedo, o que torna sua interação muito prática já que não é necessário ter a mão dispositivos secundários, como caneta digital, mouse ou teclado físico. Por outro lado, ao contrário de outros tipos de entrada de dados como o mouse, os

dedos das pessoas variam muito de tamanho fazendo com que esse tipo de entrada seja despadronizado, menos assertivo e mais suscetíveis a toques acidentais.

Esse modo de interação também remete a um dos maiores problemas relacionado a entrada de dados: a digitação. Na digitação os usuários precisam dividir continuamente a atenção entre o conteúdo que eles estão digitando e a área do teclado, teclado este pequeno e com teclas muito próximas. Sendo assim, não só a tela dos *smartphones* são menores e capazes de armazenar menos conteúdo que *desktops* e *laptops*, mas os botões e os demais elementos de interação precisam ser maiores do que aqueles em um monitor tradicional.

Mesmo na era de redes de celulares rápidas e *Wi-Fi*[1] por toda parte, a cobertura não é algo universal ou igualmente boa. É comum os usuários de *smartphones* se queixarem de problemas de conectividade. Quando a rede está lenta ou instável cada novo carregamento de página se traduz em um tempo de espera significativo ou ainda na falha de execução de uma tarefa. O contexto dinâmico e a instabilidade na conectividade deve estar em mente durante todo o processo de design dos aplicativos. Construir páginas leves, otimizar carregamentos e minimizar o tráfego constante entre servidor e cliente são estratégias importantes para evitar a frustração dos usuários e prover uma boa experiência.

Não é só de limitações que são feitos os *smartphones*, mas também de diversos recursos que, quando negligenciados, podem contribuir para uma má experiência do usuário. A câmera, o microfone e o *GPS*[2], por exemplo, são convenientemente integrados ao dispositivo e podem ser facilmente usados para facilitar a entrada e evitar a dificuldade da digitação; as notificações permitem que os usuários sejam atualizados imediatamente de eventos considerados relevantes por eles; o identificador de toque permite aos usuários se autenticarem usando sua impressão digital, ou seja, sem a necessidade de digitar senhas.

[1] Wireless Fidelity
[2] Global Positioning System

3 Engenharia de Software

Este capítulo apresenta conceitos da Engenharia de Software que foram relevantes no desenvolvimento dos componentes que complementam o guia proposto nesse trabalho. Entre os temas abordados neste capítulo está o conceito de reúso para a Engenharia de Software, o desenvolvimento de componentes na aplicação do reúso e de sua importância no contexto de aplicações para dispositivos móveis, assim como o conceito de padrões de projeto e a sua relevância no desenvolvimento de componentes de software.

3.1 Reúso

O termo "reúso de software" foi utilizado oficialmente pela primeira vez na Conferência de Engenharia de Software da NATO em 1968, conferência essa considerada o berço da Engenharia de Software. Apesar de ter sido proposto como uma estratégia de desenvolvimento na década de 60, só em 2000 o desenvolvimento de software com reúso se tornou a norma para novos sistemas de negócios (SOMMERVILLE, 2011, p. 297).

Segundo Sommerville (2011, p. 297) "a mudança para o desenvolvimento baseado em reúso foi uma resposta às exigências de menores custos de produção e manutenção de software, entregas mais rápidas de sistemas e softwares de maior qualidade". Os benefícios existentes com a adoção dessa estratégia de desenvolvimento foram listados por Sommerville (2011, p. 297) e encontram-se dispostos na Tabela 2.

A engenharia de software baseada em reúso é uma estratégia de desenvolvimento em que o reúso de softwares existentes é maximizado. Nessa estratégia os softwares a serem reutilizados podem variar bastante de tamanho, indo desde sistemas inteiros, passando por subsistemas e componentes, até simples objetos e funções.

3.2 Componentes de Software

Devido a sua natureza independente, os componentes de softwares representam um dos níveis com maior potencial para aplicação do reúso de software. Embora não exista um consenso quanto a sua definição, de modo geral as definições atribuídas aos componentes de software se baseiam em duas ideias. A ideia de um componente como sendo uma parte que forma um sistema e a ideia de um componente como sendo um serviço a ser referenciado pelo sistema. Sommerville (2011, p. 317) acredita que uma definição melhor para um componente possa ser obtida combinando as duas ideias e apresenta na Tabela 1 o conjunto de características que ele considera como essenciais para um componente. Para

Skyperski (SOMMERVILLE, 2011, p. 317) um componente de software é:

> [...] uma unidade de composição com interfaces contratualmente especificadas e apenas dependências de contexto explícitas. Um componente de software pode ser implantado de forma independente e está sujeito a ser composto por parte de terceiros. (SOMMERVILLE, 2011, p. 317)

No contexto de desenvolvimento para plataforma móvel, o uso de componentes é uma realidade. Segundo a Google (2016) um dos conceitos fundamentais sobre a estrutura de aplicativos do sistema operacional Android é que os aplicativos para Android são criados como uma combinação de componentes distintos que podem ser invocados individualmente.

Como exemplo de uso de componentes no desenvolvimento de aplicativos o CodePath (2016) mantém uma lista de componentes e bibliotecas desenvolvidos por terceiros que , segundo ele, são extremamente populares e frequentemente utilizados em diversos projetos para plataforma Android. E apesar desses componentes terem diferentes utilidades, todos têm o mesmo objetivo: diminuir o esforço do desenvolvedor.

3.3 Padrões de Projeto

Os padrões de projeto foram introduzidos na Engenharia de Software a partir das ideias apresentadas por Christopher Alexander, um arquiteto e urbanista austríaco, que sugeriu haver padrões comuns de projeto de prédios que eram intrinsecamente agradáveis e eficazes (SOMMERVILLE, 2011, p. 133). Para Sommerville (2011, p. 133) um padrão de projeto é uma descrição do problema e da essência da solução, de modo que a solução possa ser reusada em diferentes contextos. Além disso um padrão pode ser enxergado como uma descrição de conhecimento e experiência, uma solução já aprovada para um problema comum.

Segundo Gamma et al. (2000, p. 20) um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável e, de modo geral, é composto de quatro elementos essenciais: o nome do padrão, o problema, a solução e as consequências. O nome do padrão é uma referência para descrever os outros elementos em poucas palavras e facilitar a comunicação entre desenvolvedores; o problema descreve o contexto apropriado para se aplicar o padrão; a solução descreve os elementos que compõem o padrão como: os relacionamentos, responsabilidades e colaborações; e as consequências descrevem as vantagens e desvantagens do uso do padrão.

Uma vez que os padrões representam soluções já aprovadas para problemas recorrentes e representam o conhecimento abstrato da experiência de outros projetos, ao utilizá-los no desenvolvimento de componentes agregamos a eles maior confiabilidade e robustez.

Tabela 1 – Características essenciais de um componente de software

Característica	**Descrição**
Padronizado	A padronização de componentes significa que um componente usado em um processo CBSE precisa obedecer a um modelo de componentes padrão. Esse modelo pode definir as interfaces de componentes, metadados de componente, documentação, composição e implantação.
Independente	Um componente deve ser independente, deve ser possível compor e implantá-lo sem precisar usar outros componentes específicos. Nessas situações, em que o componente precisa dos serviços prestados externamente, estes devem ser explicitamente definidos em uma especificação de interface "requires" .
Passível de composição	Para um componente ser composto, todas as interações externas devem ter lugar por meio de interfaces publicamente definidas. Além disso, ele deve proporcionar acesso externo a informações sobre si próprio, como seus métodos e atributos
Implantável	Para ser implantável, um componente deve ser autocontido. Deve ser capaz de operar como uma entidade autônoma em uma plataforma de componentes que forneça uma implementação do modelo de componentes, o que geralmente significa que o componente é binário e não tem como ser compilado antes de ser implantado. Se um componente é implantado como um serviço, ele não precisa ser implantado por um usuário de um componente. Pelo contrário, é implantado pelo prestador do serviço.
Documentado	Os componentes devem ser completamente documentados para que os potenciais usuários possam decidir se satisfazem as suas necessidades. A sintaxe e, idealmente, a semântica de todas as interfaces de componentes devem ser especificadas.

Fonte: (SOMMERVILLE, 2011, p. 317)

Tabela 2 – Benefícios do reúso de software

Benefício	**Explicação**
Confiança aumentada	Os softwares reusados, experimentados e testados em sistemas em funcionamento provavelmente são mais confiáveis do que um novo software. Seus defeitos de projeto e implementação devem ser encontrados e corrigidos.
Risco de processo reduzido	O custo do software existente já é conhecido, considerando que os custos de desenvolvimento são sempre uma questão que envolve julgamento. Esse é um importante fator para o gerenciamento de projeto, pois reduz a margem de erro de estimativa de custos de projeto, o que é particularmente verdadeiro quando componentes de software relativamente grandes, como subsistemas, são reusados.
Uso eficaz de especialistas	Em vez de repetir o mesmo trabalho, especialistas em aplicações podem desenvolver softwares reusáveis que encapsulem seu conhecimento
Conformidade com padrões	Alguns padrões, como os de interface de usuário, podem ser implementados como um conjunto de componentes reusáveis. Por exemplo, se os menus em uma interface de usuário forem implementados usando componentes reusáveis, todas as aplicações apresentarão os mesmos formatos de menu para os usuários. O uso de interfaces de usuário-padrão melhora a confiança, pois os usuários cometem menos erros quando são apresentados a interfaces familiares.
Desenvolvimento acelerado	Muitas vezes, os custos gerais de desenvolvimento não são tão importantes quanto entregar um sistema ao mercado o mais rápido possível. O reúso de um software pode acelerar a produção do sistema, pois pode reduzir o tempo de desenvolvimento e validação.

Fonte: (SOMMERVILLE, 2011, p. 297)

4 Guia de Usabilidade

O objetivo deste trabalho consiste na elaboração de um guia de usabilidade, independente do sistema operacional, que incorpore um conjunto de heurísticas adequadas para os aspectos característicos dos *smartphones*. Para isso, além do guia foram desenvolvidos dois componentes que atendem às especificações do guia proposto.

Neste capítulo serão apresentados os trabalhos correlatos que nortearam esse trabalho, o percurso metodológico para a elaboração do guia, do desenvolvimento dos componentes e dos experimentos realizados, bem como os resultados obtidos.

4.1 Trabalhos Correlatos

Para fundamentar o presente trabalho foi realizada uma pesquisa sobre heurísticas de usabilidade no contexto de dispositivos móveis a fim de encontrar trabalhos com propostas de heurísticas específicas para os *smartphones*. Os trabalhos encontrados encontram-se detalhados a seguir.

No trabalho de Joyce e Lilley (2014) as heurísticas de Nielsen e Molich assim como tópicos da literatura sobre interação humano computador (IHC) foram usados como base no desenvolvimento de treze heurísticas feitas para a inspeção de aplicações móveis. Esse conjunto de heurísticas foi então submetido a avaliação de aproximadamente sessenta especialistas e pesquisadores de IHC espalhados por volta de dezoito países, o que culminou no refinamento e modificação dessas heurísticas. Apesar de uma avaliação empírica sobre esse conjunto estar pendente os resultados iniciais mostraram-se promissores visto as análises positivas do grupo de avaliadores.

No trabalho de Neto e Pimentel (2013) os autores estenderam as heurísticas de Nielsen para derivar um conjunto de heurísticas específicas para a avaliação de usabilidade de interfaces de usuário móveis. O conjunto proposto foi compilado com base em problemas identificados ao avaliar alguns aplicativos populares de dispositivos móveis que eram difíceis de associar às heurísticas de Nielsen. Para validar o conjunto de heurísticas proposto dois grupos de especialistas distintos avaliaram uma aplicação Android usando as heurísticas de Nielsen e as heurísticas propostas.

Inostroza et al. (2013) apresenta em seu trabalho um conjunto de doze heurísticas de usabilidade específicas para dispositivos móveis baseados em telas sensíveis ao toque. A metodologia utilizada para conceber esse novo conjunto de heurísticas foi composta de seis estágios: o primeiro estágio consistiu na exploração dos principais tópicos relacionados a heurísticas de usabilidade; o segundo estágio consistiu na descrição dos principais conceitos

encontrados na exploração realizada no estágio anterior; o terceiro estágio foi onde foram identificadas as características que as heurísticas deveriam ter, baseada nas heurísticas tradicionais e na análise de estudos de caso; o quarto estágio foi a especificação formal do conjunto de heurísticas proposto; o quinto estágio foi a validação desse conjunto de heurísticas por meio de experimentos, avaliações heurísticas e testes de usuários; o sexto e último estágio consistiu no refinamento das heurísticas baseado no *feedback* obtido no estágio de validação.

Tanto os trabalhos de Neto e Pimentel (2013) quanto o de Inostroza et al. (2013) apresentaram resultados que verificam a utilidade das heurísticas específicas para dispositivos móveis e um melhor desempenho na identificação de problemas de usabilidade em relação as heurísticas tradicionais. Apesar da ausência de uma validação empírica no trabalho de Joyce e Lilley (2014), as críticas positivas feitas pelos especialistas e pesquisadores da área de IHC sugerem que as heurísticas criadas são promissoras.

A Google e a Apple, desenvolvedoras dos principais sistemas operacionais para *smartphones*, disponibilizam em seus websites diretrizes para o design de interfaces dos aplicativos de seus respectivos sistemas operacionais: o Android e o iOS. Pautadas em convenções, artigos e estudos na área de IHC, essas documentações buscam através de recomendações, padrões e boas práticas, auxiliar designers e desenvolvedores na criação de aplicações capazes de prover uma melhor experiência aos seus usuários. Essas documentações são o *Material Guidelines*, da Google, e o *iOS Human Interface Guidelines*, da Apple. O *Material Guidelines* (GOOGLE, 2018) é um guia formado por um conjunto de diretrizes sobre animações, interações e componentes construído em cima do *Material Design*, uma linguagem visual desenvolvida e adotada pela Google em suas plataformas. Já o *iOS Human Interface Guidelines* (APPLE, 2018) é uma documentação formada por uma série de diretrizes que abrange boas práticas e padrões consistentes em todas as plataformas da Apple.

Apesar de contribuírem até certo ponto para o desenvolvimento de interfaces mais usáveis e nelas ser possível encontrar uma série de princípios de design e usabilidade relevantes, essas documentações são extensas e apresentam como foco a consistência dos aplicativos e a unificação da experiência dos usuários nas suas diversas plataformas.

Dessa forma, embasado nas heurísticas propostas pelos trabalhos supracitados e nos princípios de design e usabilidade dispostos no *Material Guidelines* e no *iOS Human Interface Guidelines*, o presente trabalho propõe um guia de usabilidade independente do sistema operacional tendo como foco questões de usabilidade.

4.2 Percurso Metodológico

Tendo em vista a elaboração do guia e o desenvolvimento dos componentes para auxiliar na construção de aplicativos móveis mais usáveis e na redução de esforço para tal, esse trabalho se enquadra quanto a sua natureza como pesquisa aplicada, uma vez que ele tem como objetivo gerar conhecimentos para aplicação prática direcionado a solucionar um problema específico (SILVEIRA; CÓRDOVA, 2009, p.35). Quanto aos procedimentos este se enquadra como pesquisa experimental, conforme definição de TRIVIÑOS (1987):

> O estudo experimental segue um planejamento rigoroso. As etapas de pesquisa iniciam pela formulação exata do problema e das hipóteses, que delimitam as variáveis precisas e controladas que atuam no fenômeno estudado. (TRIVIÑOS, 1987)

No que diz respeito a etapa de verificação o presente trabalho também pode ser classificado como pesquisa qualitativa e interpretativa, uma vez que a estratégia adotada para verificar a efetividade do guia e dos componentes foi uma análise qualitativa a partir da percepção do pesquisador.

Com base nas classificações supracitadas foram definidas cinco etapas para alcançar o objetivo desse trabalho, são elas: uma revisão bibliográfica, a elaboração do guia, o desenvolvimento dos componentes, a verificação e a análise dos resultados.

Na primeira etapa foi realizada uma revisão bibliográfica a respeito das heurísticas e conceitos de usabilidade para embasar a pesquisa. Em seguida foi feita a elaboração do guia a partir das heurísticas específicas para *smartphones* e dos princípios de design e usabilidade reunidos na etapa anterior. Finalizado o guia, foram selecionados dois componentes nativos do Android e alguns elementos do guia para serem incorporados por esses componentes. Ao final do desenvolvimento dos componentes e uma vez concebido o guia a etapa de verificação foi realizada. Essa verificação foi dividida em dois experimentos onde procurou-se analisar ora a efetividade do guia, ora dos componentes. Por fim teve curso uma análise dos resultados.

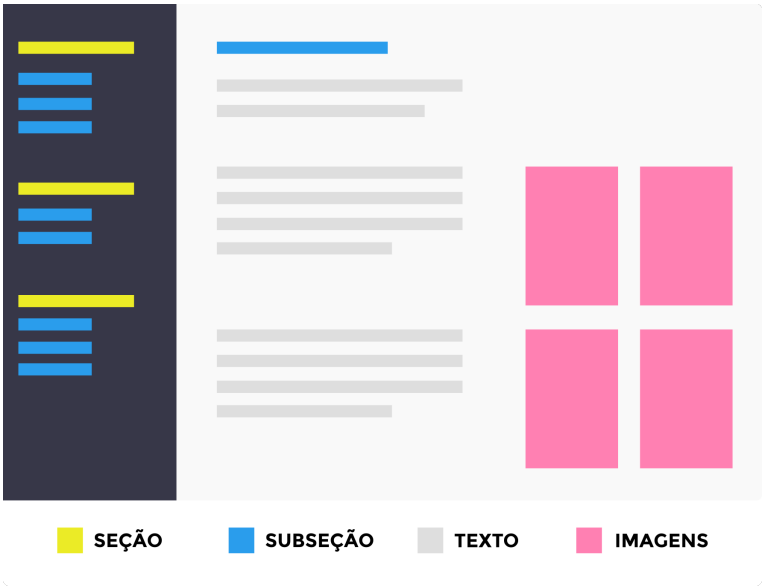
4.3 Elaboração do Guia

A elaboração do guia de usabilidade é fundamentado nas heurísticas de usabilidade propostas nos trabalhos citados na seção 4.1, nas heurísticas de Nielsen (1995) e nos guias de design de interfaces dos dois sistemas operacionais mais vendidos: iOS e Android. O guia foi disponibilizado na web e pode ser acessado através do link: <https://simasdanilo.github.io/tcc/guia>.

Conforme ilustrado na Figura 1 o guia foi estruturado em seções e subseções. As seções correspondem aos títulos das dez heurísticas de Nielsen visto suas características

genéricas, enquanto as subseções foram criadas pela necessidade de agrupar temas relacionados. Nas subseções encontram-se descritas as recomendações do guia junto às suas respectivas imagens ilustrativas.

Figura 1 – Estrutura do guia



Fonte: Elaborado pelo autor.

A primeira etapa na elaboração do guia consistiu em selecionar dentre as heurísticas específicas para dispositivos móveis propostas nos trabalhos supracitados aquelas que não tratavam de um contexto ou componente específico e relacioná-las as seções criadas. Essas heurísticas selecionadas encontram-se dispostas no Anexo A.

No passo seguinte, os princípios de design e usabilidade foram extraídos dos guias de design de interfaces e também foram relacionados às seções. As heurísticas e princípios foram então agrupados em subseções para melhor organizar o guia. Assim, as heurísticas e os princípios reunidos foram utilizados para escrever as recomendações de cada subseção do guia. Por fim, as imagens ilustrando as recomendações foram produzidas e inseridas no guia.

Em seguida encontram-se resumidas as recomendações encontradas em cada seção do guia elaborado.

1. **Visibilidade do status do sistema:** recomendações sobre como melhorar a compreensão do usuário a cerca de onde ele está, o que ele pode fazer e o que está acontecendo na tela. Nessa seção aborda-se sobre gestos na navegação, *feedback*, reconhecimento de ações, efeitos de animação e transição, processos em andamento e tratamento de estado vazios;

2. **Controle e liberdade do usuário:** recomendações sobre como fornecer ao usuário a capacidade de desfazer suas ações e sair de estados indesejados;

3. **Consistência e padrões:** recomendações sobre como implementar padrões e prover uma experiência consistente durante todas as telas e todas as interações;

4. **Prevenção de erro:** recomendações sobre como minimizar e antecipar erros, e quando requisitar confirmações para as ações do usuário;

5. **Flexibilidade e eficiência de uso:** recomendações sobre como fornecer atalhos, interações alternativas e recursos extras para facilitar na realização das tarefas, satisfazendo tanto usuários novos quanto usuários experientes;

6. **Estética e design minimalista:** recomendações a respeito dos elementos das interfaces, como o tamanho da área de toque mínimo para uma interação confiável e da utilização de cores e contrastes para destacar elementos ou identificar elementos interativos;

7. **Reconhecer, diagnosticar e recuperar erros:** recomendações de como tratar e abordar erros, além de como tratar e informar problemas de conectividade, visto que uma grande parcela de aplicações são fortemente dependentes da conectividade;

8. **Ajuda e documentação:** recomendações sobre como moldar a primeira experiência do usuário com o aplicativo e como apresentar os principais recursos e funcionalidades do aplicativo aos novos usuários;

4.4 Desenvolvimento dos Componentes

Para auxiliar os desenvolvedores na construção de suas aplicações, numa perspectiva de reúso, é possível automatizar as recomendações do guia através da utilização de componentes que incorporam essas recomendações. Como demonstração dessa abordagem foram desenvolvidos dois componentes, para o sistema operacional Android, que incorporam algumas das recomendações dispostas no guia. A implementação dos componentes encontram-se nos Apêndices A e B, e suas documentações podem ser acessadas pelo link: <https://simasdanilo.github.io/tcc/componentes>.

O primeiro passo no desenvolvimento desses componentes foi a escolha dos componentes de interface visual nativos do Android que seriam utilizados para incorporar as recomendações do guia. Os componentes escolhidos foram o **Recyclerview** e o **ImageButton**. O **RecyclerView** é um componente utilizado para exibir listas e foi escolhido pela fato da lista ser a forma de apresentação mais comumente utilizada para representar dados e assim estar presente na maioria dos aplicativos. O **ImageButton** é um componente utilizado para exibir um botão com uma imagem e foi escolhido pelo fato dos botões com ícones serem um dos principais elementos de interação encontrados nas aplicações.

4.4.1 Componente LiloRecyclerView

O **RecyclerView** é um componente visual formado por diferentes componentes que juntos trabalham para exibir um grande conjunto de dados em forma de lista. Dentre os componentes que o compõem destaca-se o **Adapter**, uma subclasse do **RecyclerView** que fornece um vínculo entre um conjunto de dados específico e os elementos visuais de cada item da lista.

Quando uma lista não contém itens ela se encontra no chamado estado vazio. Segundo o guia, na seção "Visibilidade e status do sistema" , os estados vazios ocorrem quando o conteúdo de um elemento não pode ser exibido e apesar desses estados não serem típicos eles devem ser projetados para evitar a confusão do usuário. É possível evitar essa confusão exibindo conteúdos alternativos ou comunicando ao usuário a ausência de dados quando a lista está vazia. A Figura 2 mostra um exemplo de um aplicativo de notificações que utiliza uma imagem não interativa e um texto para comunicar a ausência de notificações, ou seja, o estado vazio.

Figura 2 – Exemplo de uma aplicação que trata o estado vazio

Fonte: Elaborado pelo autor.

Para fazer com que o **RecyclerView** tratasse o estado vazio como recomendado pelo guia, foi desenvolvido o **LiloRecyclerView**, um componente que estende o **RecyclerView** e adiciona a ele a capacidade de definir uma imagem de plano de fundo para ser exibida no seu estado vazio. Dessa forma, quando não há itens para serem apresentados na lista o plano de fundo definido é exibido no lugar.

A implementação do **LiloRecyclerView** foi feita com a utilização do padrão de projeto **Observer**. Segundo Gamma et al. (2000, p.274) esse padrão define uma dependência um-para-muitos entre objetos, de forma que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente. A solução proposta por esse padrão gira em torno de duas classes: o **Subject** e o **Observer**. O **Subject** registra uma quantidade qualquer de observadores e os notifica toda vez que ele sofre uma mudança em seu estado. O **Observer**, por sua vez, é notificado pelo **Subject** através de uma interface de atualização.

Como pode ser visto no Código 4.1, quando um **Adapter** é atribuído ao **LiloRecyclerView** o **AdapterDataObserver** é registrado a ele, assim, toda vez que há uma mudança no conjunto de dados vinculado ao **Adapter** o **AdapterDataObserver** é notificado e o método **updateBackground** é chamado. Esse método, por sua vez, verifica se a lista encontra-se no estado vazio e caso afirmativo modifica o plano de fundo da lista para o da imagem pré-definida para esse estado. A definição dessa imagem pode ser feita pelo desenvolvedor através do método **setEmptyStateDrawable** como mostra o Código 4.2.

Código 4.1 – Solução utilizando o padrão Observer

```
1    private Drawable emptyStateDrawable;
2    private AdapterDataObserver adapterDataObserver = new
    AdapterDataObserver() {
3        @Override
4        public void onChanged() {
5            super.onChanged();
6            updateBackground();
7        }
8    };
9
10   @Override
11   public void setAdapter(RecyclerView.Adapter adapter) {
12       Adapter currentAdapter = this.getAdapter();
13
14       if (currentAdapter != null)
15           currentAdapter.unregisterAdapterDataObserver(
    adapterDataObserver);
16
17       if (adapter != null)
18           adapter.registerAdapterDataObserver(adapterDataObserver);
19
20       super.setAdapter(adapter);
21       this.updateBackground();
22   }
23
24    /**
```

```
25      * Updates the current background
26      */
27     private void updateBackground() {
28         Adapter adapter = this.getAdapter();
29 
30         if (emptyStateDrawable == null || adapter == null)
31             return;
32 
33         int itemCount = adapter.getItemCount();
34         boolean showEmptyState = itemCount == 0;
35 
36         if (showEmptyState)
37             this.setBackground(this.emptyStateDrawable);
38         if (!showEmptyState)
39             this.setBackground(null);
40     }
```

Código 4.2 – Método para definir a imagem do estado vazio

```
1     /**
2      * Sets the empty state drawable
3      * @param emptyStateDrawable drawable to show when in empty state
4      */
5     public void setEmptyStateDrawable(Drawable emptyStateDrawable) {
6         this.emptyStateDrawable = emptyStateDrawable;
7         this.updateBackground();
8     }
```

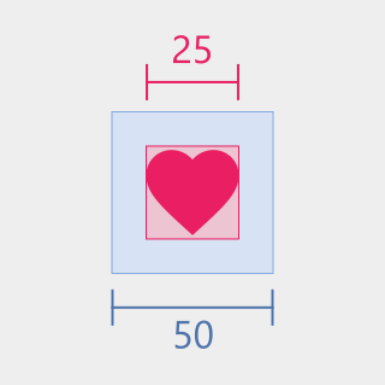
4.4.2 Componente LiloImageButton

O tamanho da área de toque de um elemento visual corresponde a área que responde ao toque do usuário e está além dos limites visuais desse elemento. Como exemplificado na Figura 3, um ícone pode ter vinte e cinco unidades de medida de altura e largura quando na verdade sua área de toque compreende cinquenta unidades de medida de altura e largura.

Segundo o guia, na seção “Estética e Design Minimalista”, o tamanho mínimo de área de toque recomendado para elementos interativos para garantir uma interação confiável , como é o caso do componente **ImageButton**, é de aproximadamente sete a dez milímetros. Em se tratando do sistema operacional Android essa área corresponde a cerca de cinquenta pixels independentes de densidade (dpi). Dpi é uma unidade de medida independente de resolução utilizada pelo Android para dar suporte a diferentes densidades de telas.

Para que o **ImageButton** atendesse a recomendação do tamanho mínimo da área de toque proposta pelo guia foi desenvolvido o **LiloImageButton**, um componente que

Figura 3 – Tamanho da área de toque



Fonte: Elaborado pelo autor.

estende o **ImageButton** e adiciona a ele a funcionalidade de garantir que as dimensões do componente sempre respeitem o tamanho mínimo recomendado. A implementação dessa funcionalidade se dá através da sobreescrita do método **onMeasure** do **ImageButton**, responsável por medir e definir sua altura e largura. Como observado no Código 4.3 o método sobreescrito verifica se as dimensões da imagem do componente são menor do que as dimensões recomendadas, caso afirmativo ele adiciona a diferença dessas dimensões como um espaçamento interno do elemento afim de que a área de toque do componente atinja o tamanho recomendado.

Código 4.3 – Sobreescrita do método onMeasure

```
9     @Override
10    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
11        Drawable drawable = this.getDrawable();
12
13        if (drawable != null) {
14            int drawableHeight = this.getDrawable().getIntrinsicHeight();
15            int drawableWidth = this.getDrawable().getIntrinsicWidth();
16
17            int paddingHorizontal = 0;
18            int paddingVertical = 0;
19
20            int minTouchTargetWidthDp = this.getMinTouchTargetWidth();
21            int minTouchTargetHeightDp = this.getMinTouchTargetHeight();
22
23            int minTouchTargetWidthPx = this.convertDpToPx(
    minTouchTargetWidthDp);
24            int minTouchTargetHeightPx = this.convertDpToPx(
    minTouchTargetHeightDp);
25
26            if (drawableWidth < minTouchTargetWidthPx) {
27                paddingHorizontal = Math.round((minTouchTargetWidthPx -
    drawableWidth) / 2f);
```

```
28              }
29
30              if (drawableHeight < minTouchTargetHeightPx) {
31                  paddingVertical = Math.round((minTouchTargetHeightPx -
    drawableHeight) / 2f);
32              }
33
34              if (paddingVertical != 0 || paddingHorizontal != 0)
35                  this.modifyPadding(paddingHorizontal, paddingVertical);
36          }
37
38          super.onMeasure(widthMeasureSpec, heightMeasureSpec);
39      }
```

Ainda segundo o guia, a depender do público a que se destina o aplicativo, pode ser apropriado usar áreas de toque maiores que as recomendadas para acomodar um maior espectro de usuários, como por exemplo, crianças com problemas de desenvolvimento de habilidades motoras. Tendo isso em mente, o **LiloImageButton** foi projetado com dois métodos, **getMinTouchTargetWidth** e **getMinTouchTargetHeight**, que quando sobreescritos permitem o desenvolvedor aumentar o tamanho da área de toque mínima definida no componente. Os Códigos 4.4 e 4.5 apresentam, respectivamente, a definição desses métodos e um exemplo de um componente que estende o **LiloImageButton**.

Código 4.4 – Métodos para sobreescrever o tamanho mínimo recomendado

```
40      /**
41       * Returns the min touch target width. The default implementation
42       * of this method returns {@link #MIN_TOUCH_TARGET_SIZE}.
43       * @return min touch target width, in density-independent pixel (dp)
44       */
45      protected int getMinTouchTargetWidth(){
46          return MIN_TOUCH_TARGET_SIZE;
47      }
48
49      /**
50       * Returns the min touch target height. The default implementation
51       * of this method returns {@link #MIN_TOUCH_TARGET_SIZE}.
52       * @return min touch target height, in density-independent pixel (dp)
53       */
54      protected int getMinTouchTargetHeight(){
55          return MIN_TOUCH_TARGET_SIZE;
56      }
```

Código 4.5 – Exemplo de um componente estendendo o LiloImageButton

```
57 import com.simas.danilo.lilo.component.LiloImageButton;
58
```

```
59 public class MyLiloImageButton extends LiloImageButton {
60
61     public static final int MY_MIN_TOUCH_TARGET_SIZE = 80;
62
63     @Override
64     protected int getMinTouchTargetHeight() {
65         return MY_MIN_TOUCH_TARGET_SIZE;
66     }
67
68     @Override
69     protected int getMinTouchTargetWidth() {
70         return MY_MIN_TOUCH_TARGET_SIZE;
71     }
72 }
```

4.5 Experimentos

Como descrito na Seção 4.2, foram realizados dois experimentos para verificar a efetividade do guia e dos componentes desenvolvidos. No primeiro experimento foi feito uma avaliação das interfaces de três aplicativos a partir das recomendações do guia proposto, a fim de mostrar a possibilidade de identificar problemas de usabilidade em aplicações já construídas. No segundo experimento uma aplicação foi desenvolvida em dois cenários: ora com componentes nativos do sistema operacional Android, ora com o guia e os componentes criados, a fim de verificar a possibilidade de automatizar as recomendações do guia através da utilização de componentes que incorporam essas recomendações.

Para a realização do primeiro experimento foram selecionados três aplicativos populares da área de mobilidade urbana contendo entre cinquenta mil e um milhão de instalações na loja de aplicativos da Google, são eles: o **Bike Itaú**, o **CittaMobi** e o **KIM Recarga**. O **Bike Itaú** é um aplicativo que auxilia a encontrar as estações de bicicletas espalhadas pela cidade e a alugar essas bicicletas; o **CittaMobi** é um aplicativo que informa sobre rotas e previsões do tempo de chegada dos ônibus; e o **KIM Recarga** é um aplicativo para gerenciamento de cartões de transporte.

Dentre o universo de recomendações dispostas no guia, dez foram selecionadas para fazer parte do experimento, onde procurou-se incluir nessa seleção as recomendações incorporadas pelos componentes desenvolvidos e descritos na Seção 4.4. Uma vez selecionados os aplicativos e definido o conjunto de recomendações, foi realizada uma avaliação das interfaces de cada um dos aplicativos a fim de verificar se os elementos dessas interfaces atendiam ao conjunto de recomendações. Este conjunto de recomendações é apresentado em seguida.

- **R1:** Forneça uma navegação intuitiva e previsível de forma que os seus usuários saibam onde estão e o que eles podem fazer;
- **R2:** Crie layouts claros destacando as ações e informações principais de cada tela;
- **R3:** Forneça *feedback* em resposta a cada ação do usuário e sobre qualquer tarefa em andamento;
- **R4:** Deixe claro quando um carregamento está ocorrendo, uma vez que uma tela em branco ou estática pode parecer que seu aplicativo está travado;
- **R5:** Apesar de atípico o estado vazio deve ser projetado para evitar a frustração e confusão do usuário;
- **R6:** Qualquer elemento na tela passível de toque ou interação deve ser suficientemente grande para uma interação confiável;
- **R7:** Identifique os elementos interativos adequadamente. As pessoas devem poder contar de relance o que um elemento faz;
- **R8:** Use uma cor contrastante para informar estados de erro, de preferência uma cor de tonalidade mais quente;
- **R9:** Mantenha os componentes no mesmo local e com o mesmo padrão visual ao longo de toda a interação. Funcionalidades semelhantes devem ser realizadas por interações similares;
- **R10:** Sempre que possível, use controles de navegação padrão uma vez que os usuários já estão familiarizados com esses controles e sabem intuitivamente como interagir com eles;

Para a realização do segundo experimento primeiramente foi idealizado um aplicativo que utiliza-se dos dois componentes desenvolvidos. Este aplicativo idealizado resume-se a uma lista de horários onde o usuário pode adicionar o horário atual e removê-lo da lista. Feito isso, foi realizado uma análise do desenvolvimento desse aplicativo em dois cenários distintos. No primeiro cenário o aplicativo foi desenvolvido utilizando apenas os componentes nativos do Android, o **RecyclerView** e o **ImageButton**. Já no segundo cenário o aplicativo foi desenvolvido com base nas recomendações do guia e fazendo uso dos componentes desenvolvidos, o **LiloRecyclerView** e o **LiloImageButton**.

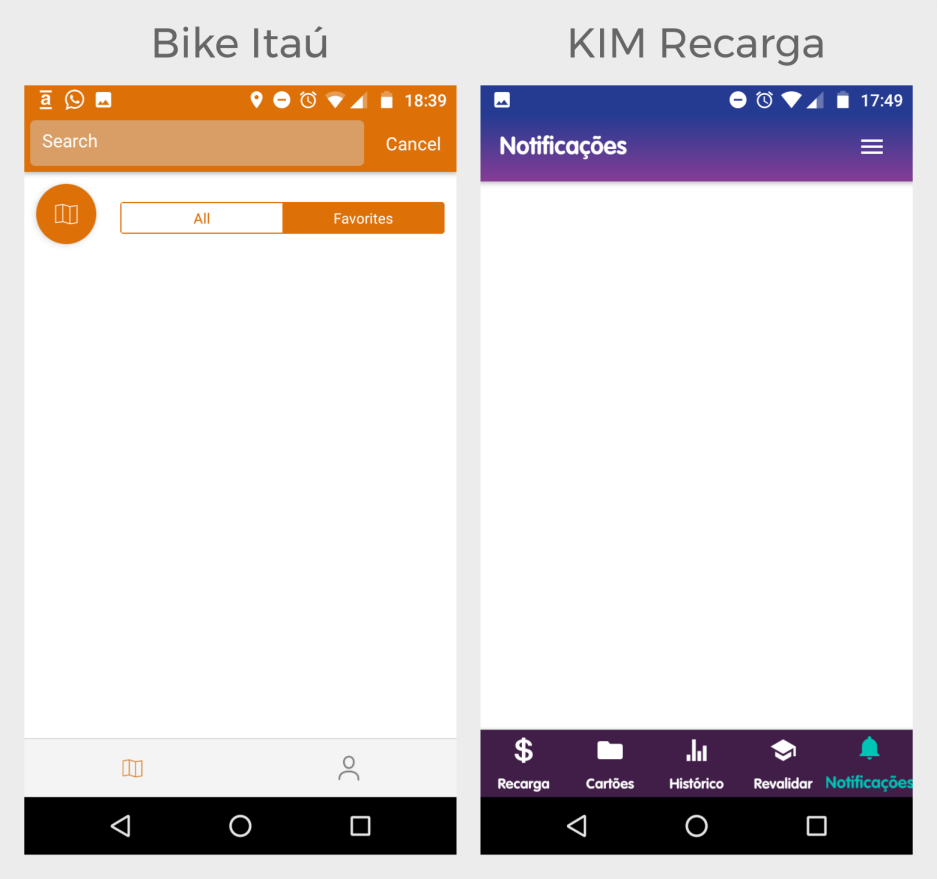
4.6 Análise dos Resultados

No que se refere ao primeiro experimento, três aplicativos tiveram suas interfaces avaliadas a partir das recomendações selecionadas, como descrito na Seção 4.5. Dos

problemas de usabilidade identificados por não atender às recomendações três foram destacados e são apresentados a seguir.

O primeiro problema de usabilidade se refere ao não atendimento da recomendação R5 que diz: "Apesar de atípico o estado vazio deve ser projetado para evitar a frustração e confusão do usuário". Na Figura 4 observa-se que tanto o aplicativo **Bike Itaú** na tela que mostra as estações favoritas do usuário, quanto o aplicativo **KIM Recarga** na tela de notificações, apresentam um espaço vazio na ausência de itens na lista, o que remete a sensação de frustração e confusão mencionada na recomendação.

Figura 4 – Telas dos aplicativos que não atenderam a recomendação R5



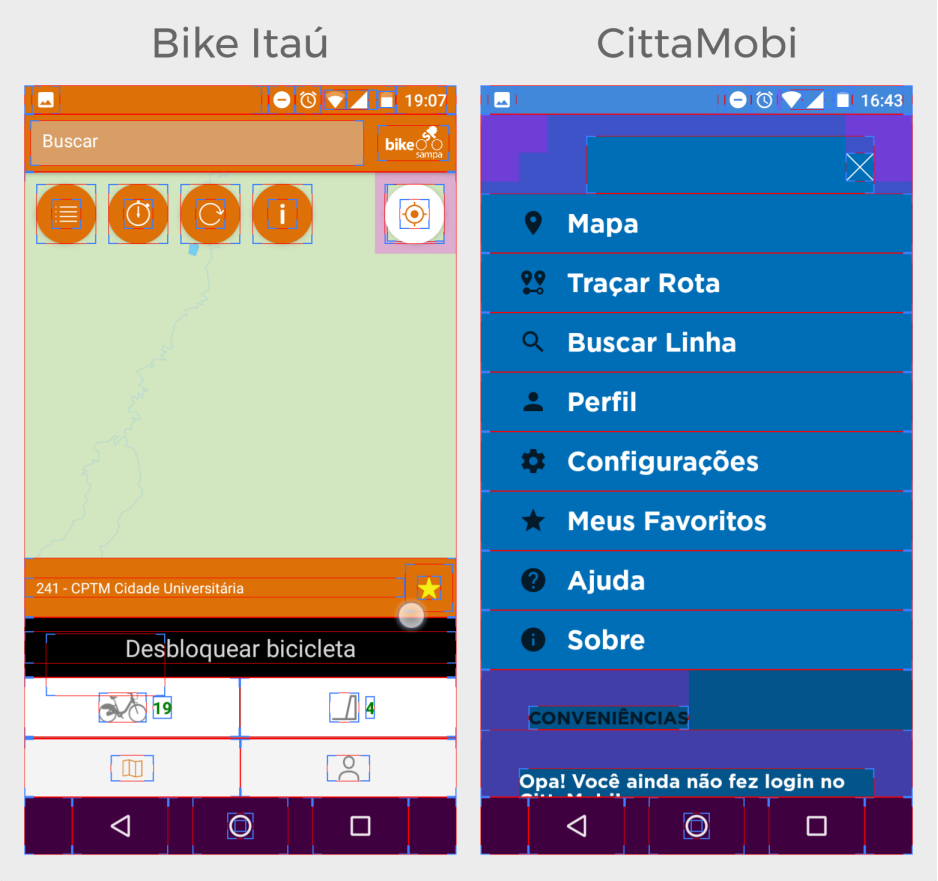
Fonte: Elaborado pelo autor

O segundo problema de usabilidade refere-se ao não atendimento da recomendação R6 que diz: "Qualquer elemento na tela passível de toque ou interação deve ser suficientemente grande para uma interação confiável". Na Figura 5 observa-se que tanto o aplicativo **Bike Itaú** na tela de visualizar a estação, quanto o aplicativo **CittaMobi** no seu menu de navegação, possuem ícones interativos com um tamanho inferior ao ideal para uma interação confiável. Esse problema foi confirmado no manuseio do aplicativo onde observou-se uma dificuldade real em interagir com os ícones.

O terceiro problema de usabilidade refere-se ao não atendimento da recomendação R7 que diz: "Identifique os elementos interativos adequadamente. As pessoas devem poder

contar de relance o que um elemento faz". Na Figura 6 observa-se que todos os três aplicativos apresentam elementos interativos visualmente semelhantes aos elementos não interativos ou dificilmente distinguíveis, e por isso, existe uma sensação de incerteza quando se interage com os elementos das telas.

Figura 5 – Telas dos aplicativos que não atenderam a recomendação R6



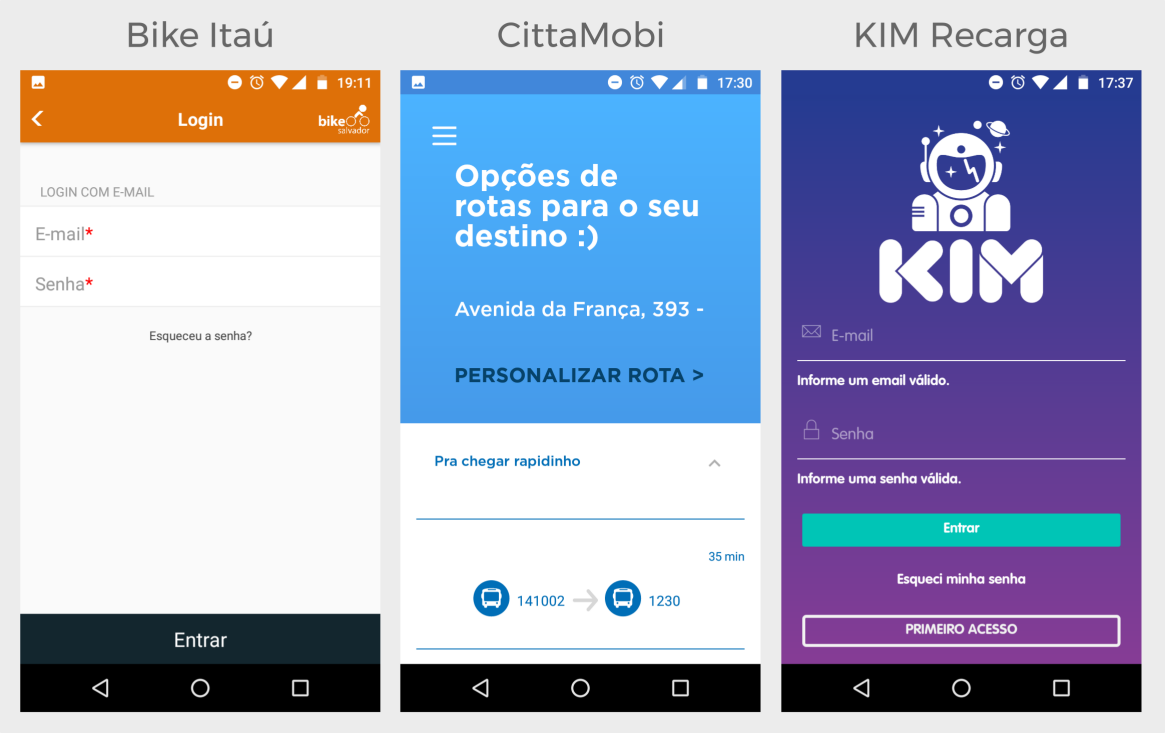
Fonte: Elaborado pelo autor.

Ao final desse experimento a Tabela 3 foi sintetizada relacionando os aplicativos quanto ao atendimento das recomendações. Na tabela a sigla "NA" significa "Não atende" , enquanto a letra "A" significa "Atende" . O número de recomendações que não foram atendidas pelos aplicativos reforçam a afirmação de que muitos aplicativos ainda apresentam problemas de usabilidade e falham em proporcionar uma boa experiência aos usuários. Além disso, as recomendações incorporadas pelos componentes desenvolvidos apresentam-se relevantes visto que dois dentre os três aplicativos falharam em atendê-las.

Apesar do seu uso ser idealizado na etapa de concepção e de prototipação de aplicativos, o guia elaborado nesse trabalho mostrou-se efetivo na identificação de problemas de usabilidade de aplicativos. Acredita-se que a utilização do guia nos estágios iniciais de concepção de um aplicativo é o cenário ideal, pois previne-se que erros de usabilidade sejam desenvolvidos e por consequência cheguem ao usuários final, evitando com isso o retrabalho por parte dos desenvolvedores e a frustração por parte dos usuários.

No que se refere ao segundo experimento, dois aplicativos foram produzidos como resultado dos cenários descritos na seção 4.5. Enquanto o aplicativo do primeiro cenário foi desenvolvido com o uso dos componentes nativos do Android, o segundo cenário teve seu aplicativo desenvolvido com o uso dos componentes criados nesse trabalho.

Figura 6 – Telas dos aplicativos que não atenderam a recomendação R7



Fonte: Elaborado pelo autor.

Na Figura 7 estão apresentadas as telas dos aplicativos quando a lista de horários não contém itens para exibir e, por isso, encontra-se no estado vazio. É possível observar nessa figura que a tela do aplicativo 1, por utilizar o componente nativo do Android, não trata automaticamente esse estado vazio e nem fornece um método para tal.

Na Figura 8 são apresentadas as telas dos aplicativos em que a lista de horários exibe um elemento. Com o auxílio da visão dos limites do desenho, proporcionada pelo sistema operacional Android, é possível visualizar os limites dos elementos das telas. A partir dessa visão é possível notar que apesar das telas serem praticamente iguais, o botão de descartar o horário representado pelo ícone "X" na tela do aplicativo 1, por utilizar-se do componente nativo do Android, não faz tratamento da sua área de toque, e por conta disso, tem sua interação comprometida.

No ponto de vista das aplicações geradas nos dois cenários, a aplicação gerada com os componentes nativos apresentaram os problemas de usabilidade que já eram esperados, uma vez que esses componentes não incorporam as recomendações do guia. Já a aplicação produzida com os componentes criados não apresentaram os mesmos problemas de usabilidade, uma vez que os componentes criados foram assertivos na incorporação das

recomendações do guia.

No ponto de vista de desenvolvedor o desenvolvimento da aplicação com os componentes criados foi considerado prático, enquanto o **LiloRecyclerView** permitiu tratar o estado vazio da lista de horários definindo uma imagem através da chamada de um método, a utilização do **LiloImageButton** permitiu a adequação do tamanho da área de toque de modo transparente, sem a chamada de um método.

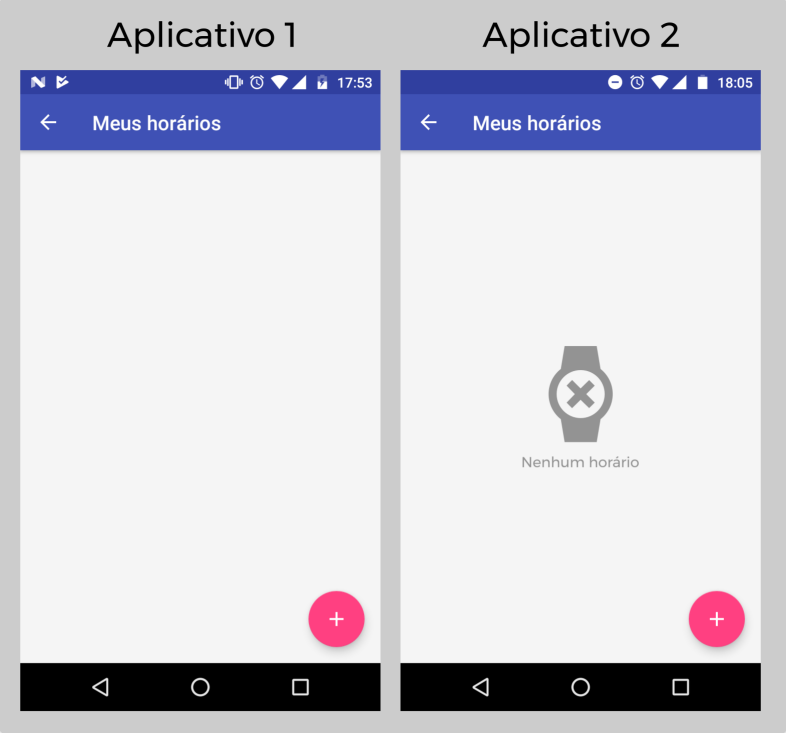
Dessa forma, é perceptível que os conhecimentos encapsulados nos componentes proveniente das recomendações do guia, fundamentados em heurísticas de usabilidade específicas para *smartphones*, e das soluções implementadas, favorecem o desenvolvimento de aplicações mais usáveis e contribuem para redução do esforço na hora de construir aplicações que atendam às recomendações do guia.

Tabela 3 – Atendimento dos aplicativos às recomendações

Recomendações/Aplicativos	**Bike Itaú**	**CittaMobi**	**KIM Recarga**
R1	NA	NA	A
R2	NA	NA	A
R3	A	A	NA
R4	NA	NA	A
R5	NA	A	NA
R6	NA	NA	A
R7	NA	NA	NA
R8	A	A	NA
R9	A	NA	A
R10	NA	NA	A

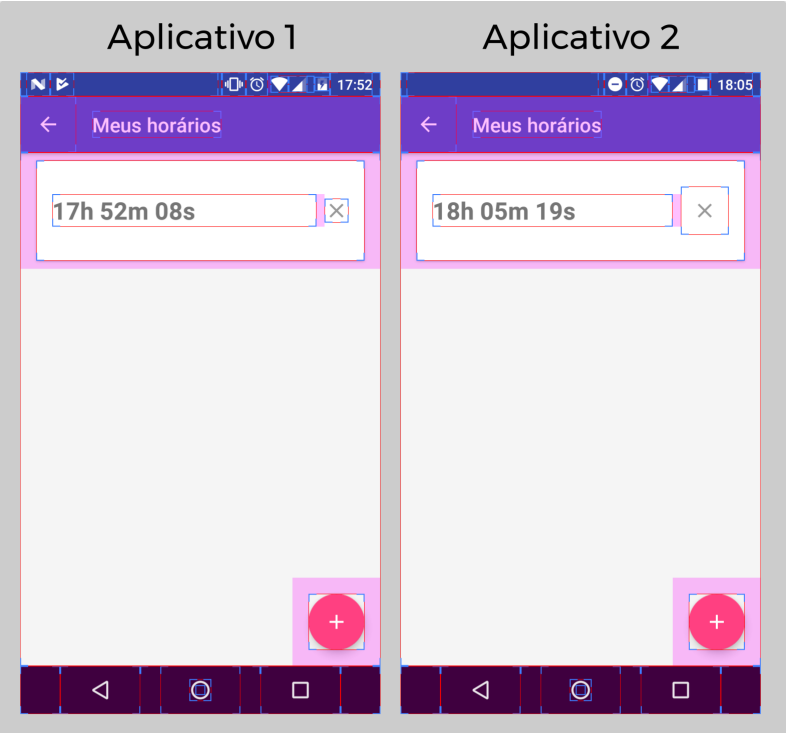
Fonte: Elaborado pelo autor.

Figura 7 – Telas dos aplicativos quando a lista de horários está vazia



Fonte: Elaborado pelo autor.

Figura 8 – Telas dos aplicativos com um horário sendo exibido na lista



Fonte: Elaborado pelo autor.

5 Considerações Finais

Este trabalho apresentou a elaboração de um guia de usabilidade independente de sistema operacional e fundamentado em heurísticas de usabilidade específicas para os *smartphones*, além do desenvolvimento de dois componentes que incorporam as recomendações desse guia proposto.

O desenvolvimento desse trabalho foi motivado pela possibilidade de apoiar desenvolvedores no processo de construção de aplicações com melhor usabilidade e na redução de esforço de testes e correções de problemas de interação dos usuários com as suas aplicações; e em uma menor esfera, pela possibilidade de contribuir para discussões sobre heurísticas de usabilidade específicas e de componentes que incorporam essas heurísticas.

A metodologia adotada responsável por determinar o percurso desse projeto permitiu alcançar plenamente os objetivos especificados, e os resultados obtidos mostraram-se satisfatórios e promissores. A partir da análise dos resultados foi possível concluir que o guia produzido apresentou-se eficaz em prever e identificar problemas de usabilidade característicos dos *smartphones*. A criação de componentes que incorporam recomendações do guia, e por isso tocam em questões de usabilidade, também foi considerada efetiva e representou uma abordagem positiva e promissora como forma de auxiliar no desenvolvimento de aplicações mais usáveis.

Como trabalhos futuros, sugere-se que seja feita uma validação do guia e dos componentes criados tendo como parte dessa validação a percepção dos usuários finais e a inserção de uma abordagem quantitativa. Sugere-se também o aumento na quantidade de componentes desenvolvidos e o desenvolvimento de componentes para outros sistemas operacionais.

Referências

APPLE. *iOS Human Interface Guidelines*. 2018. Disponível em: <https://developer.apple.com/ios/human-interface-guidelines/overview/themes/>. Citado na página 22.

BUDIU, R. *Mobile User Experience: Limitations and Strengths*. 2015. Disponível em: <http://bit.ly/1REinpX/>. Citado na página 15.

CODEPATH. *Android: Must Have Libraries*. 2016. Disponível em: <http://bit.ly/1twylbS/>. Citado na página 18.

EMARKETER. *App Downloads Will Double in Next Four Years*. 2014. Disponível em: <http://bit.ly/1fBmoYC/>. Citado na página 9.

EMARKETER. *The Mobile App Environment Is More Challenging than Ever*. 2016. Disponível em: <http://bit.ly/2dagYZz/>. Citado na página 9.

GAMMA, E. et al. ***Padrões de Projeto***: Soluções reutilizáveis de software orientado a objetos. 9. ed. Porto Alegre: Bookman, 2000. Citado 2 vezes nas páginas 18 e 27.

GOOGLE. *Guia dos Desenvolvedores: Introdução ao Android*. 2016. Disponível em: <http://bit.ly/2dyiLGF/>. Citado na página 18.

GOOGLE. *Material Guidelines*. 2018. Disponível em: <https://material.io/design/guidelines-overview/>. Citado na página 22.

GOOGLE; IPSOS. *Mobile App Marketing Insights: How Consumers Really Find and Use Your Apps*. 2015. Disponível em: <http://bit.ly/1EXl3ZA/>. Citado na página 9.

GOVE, J. *Principles of Mobile App Design: Introduction*. 2016. Disponível em: <http://bit.ly/1XFXprf/>. Citado na página 9.

INOSTROZA, R. et al. Usability heuristics for touchscreen-based mobile devices: Update. 11 2013. Citado 2 vezes nas páginas 21 e 22.

JOYCE, G.; LILLEY, M. Towards the development of usability heuristics for native smartphone mobile applications. In: MARCUS, A. (Ed.). *Design, User Experience, and Usability. Theories, Methods, and Tools for Designing the User Experience*. Cham: Springer International Publishing, 2014. p. 465–474. ISBN 978-3-319-07668-3. Citado 2 vezes nas páginas 21 e 22.

NETO, O.; PIMENTEL, M. da G. Heuristics for the assessment of interfaces of mobile devices. p. 93–96, 11 2013. Citado 2 vezes nas páginas 21 e 22.

NIELSEN, J. *10 Usability Heuristics for User Interface Design*. 1995. Disponível em: <https://bit.ly/1OmnAgZ>. Citado 3 vezes nas páginas 9, 13 e 23.

NIELSEN, J. *Usability 101: Introduction to Usability*. 2012. Disponível em: <http://bit.ly/1OOHO8T/>. Citado na página 11.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces. ACM, New York, NY, USA, p. 249–256, 1990. Disponível em: <http://doi.acm.org/10.1145/97243.97281>. Citado na página 13.

NORMAN, D. A. *The Design of Everyday Things*. New York, NY, USA: Basic Books, Inc., 2002. ISBN 9780465067107. Citado na página 12.

PREECE, J.; ROGERS, Y.; SHARP, H. *Design de Interação*: Além da interação homem-computador. 3. ed. São Paulo: Bookman, 2005. Citado 4 vezes nas páginas 11, 12, 13 e 15.

SALAZAR, L. et al. A systematic literature review on usability heuristics for mobile phones. v. 5, p. 50–61, 08 2015. Citado na página 10.

SILVEIRA, D. T.; CÓRDOVA, F. P. *Métodos de Pesquisa*. 3. ed. Porto Alegre: UFRGS Editora, 2009. Citado na página 23.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Education, Inc., 2011. Citado 4 vezes nas páginas 17, 18, 19 e 20.

STATISTA. *Number of apps available in leading app stores 2016*. 2016b. Disponível em: <http://bit.ly/2dycQpS/>. Citado na página 9.

TRIVIÑOS, A. *Introdução à pesquisa em ciências sociais: a pesquisa qualitativa em educação : o positivismo, a fenomenologia, o Marxismo*. [S.l.]: Atlas, 1987. ISBN 9788522402731. Citado na página 23.

Apêndices

APÊNDICE A – Componente LiloRecyclerView

```java
package com.simas.danilo.lilo.component;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.support.v7.widget.RecyclerView;
import android.util.AttributeSet;

public class LiloRecyclerView extends RecyclerView {
    //———————  Attributes ———————
    private Drawable emptyStateDrawable;
    private AdapterDataObserver adapterDataObserver = new
    AdapterDataObserver() {
        @Override
        public void onChanged() {
            super.onChanged();
            updateBackground();
        }
    };
    //——————— Constructor ———————
    public LiloRecyclerView(Context context) {
        super(context);
    }
    public LiloRecyclerView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
    public LiloRecyclerView(Context context, AttributeSet attrs, int
    defStyle) {
        super(context, attrs, defStyle);
    }
    // ——————— @Override ———————
    @Override
    public void setAdapter(RecyclerView.Adapter adapter) {
        Adapter currentAdapter = this.getAdapter();

        if (currentAdapter != null)
            currentAdapter.unregisterAdapterDataObserver(
    adapterDataObserver);

        if (adapter != null)
            adapter.registerAdapterDataObserver(adapterDataObserver);
```

```
38
39          super.setAdapter(adapter);
40          this.updateBackground();
41      }
42      // ———— EmptyState Methods ————
43
44      /**
45       * Sets the empty state drawable
46       * @param emptyStateDrawable drawable to show when in empty state
47       */
48      public void setEmptyStateDrawable(Drawable emptyStateDrawable) {
49          this.emptyStateDrawable = emptyStateDrawable;
50          this.updateBackground();
51      }
52
53      /**
54       * Updates the current background
55       */
56      private void updateBackground() {
57          Adapter adapter = this.getAdapter();
58
59          if (emptyStateDrawable == null || adapter == null)
60              return;
61
62          int itemCount = adapter.getItemCount();
63          boolean showEmptyState = itemCount == 0;
64
65          if (showEmptyState)
66              this.setBackground(this.emptyStateDrawable);
67          if (!showEmptyState)
68              this.setBackground(null);
69      }
70  }
```

APÊNDICE B – Componente LiloImageButton

```java
package com.simas.danilo.lilo.component;
import android.content.Context;
import android.graphics.drawable.Drawable;
import android.support.v7.widget.AppCompatImageButton;
import android.util.AttributeSet;

public class LiloImageButton extends AppCompatImageButton {
    // ——————— Attributes ———————
    // default min touch target size, in density-independent pixel (dp)
    public static final int MIN_TOUCH_TARGET_SIZE = 50;
    // ——————— Constructors ———————
    public LiloImageButton(Context context){
        super(context);
    }
    public LiloImageButton(Context context, AttributeSet attrs){
        super(context, attrs);
    }
    public LiloImageButton(Context context, AttributeSet attrs, int
    defStyleAttr){
        super(context,attrs, defStyleAttr);
    }
    // ——————— Customizable Methods ———————
    /**
     * Returns the min touch target width. The default implementation
     * of this method returns {@link #MIN_TOUCH_TARGET_SIZE}.
     * @return min touch target width, in density-independent pixel (dp)
     */
    public int getMinTouchTargetWidth(){
        return MIN_TOUCH_TARGET_SIZE;
    }
    /**
     * Returns the min touch target height. The default implementation
     * of this method returns {@link #MIN_TOUCH_TARGET_SIZE}.
     * @return min touch target height, in density-independent pixel (dp)
     */
    public int getMinTouchTargetHeight(){
        return MIN_TOUCH_TARGET_SIZE;
    }
    // ——————— @Override ———————
    @Override
```

```
40      protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
41          Drawable drawable = this.getDrawable();
42
43          if (drawable != null) {
44              int drawableHeight = this.getDrawable().getIntrinsicHeight();
45              int drawableWidth = this.getDrawable().getIntrinsicWidth();
46
47              int paddingHorizontal = 0;
48              int paddingVertical = 0;
49
50              int minTouchTargetWidthDp = this.getMinTouchTargetWidth();
51              int minTouchTargetHeightDp = this.getMinTouchTargetHeight();
52
53              int minTouchTargetWidthPx = this.convertDpToPx(
    minTouchTargetWidthDp);
54              int minTouchTargetHeightPx = this.convertDpToPx(
    minTouchTargetHeightDp);
55
56              if (drawableWidth < minTouchTargetWidthPx) {
57                  paddingHorizontal = Math.round((minTouchTargetWidthPx -
    drawableWidth) / 2f);
58              }
59
60              if (drawableHeight < minTouchTargetHeightPx) {
61                  paddingVertical = Math.round((minTouchTargetHeightPx -
    drawableHeight) / 2f);
62              }
63
64              if (paddingVertical != 0 || paddingHorizontal != 0)
65                  this.modifyPadding(paddingHorizontal, paddingVertical);
66          }
67
68          super.onMeasure(widthMeasureSpec, heightMeasureSpec);
69      }
70      // ------- General Methods -------
71
72      private int convertDpToPx(int dp){
73          // Get the screen's density scale
74          final float dpi = this.getResources().getDisplayMetrics().density;
75          // Convert the dps to pixels, based on density scale
76          int px = (int) (dp * dpi + 0.5f);
77          return px;
78      }
79
80      private void modifyPadding(int paddingHorizontal, int paddingVertical)
    {
81          this.setPadding(
```

```
82                      paddingHorizontal,
83                      paddingVertical,
84                      paddingHorizontal,
85                      paddingVertical);
86      }
87  }
```

Anexos

ANEXO A – Heurísticas Específicas para Dispositivos Móveis Selecionadas

Usability Heuristics for Touchscreen-based Mobile Devices

Rodolfo Inostroza, Cristian Rusu, Silvana Roncagliolo e Virginica Rusu

1. **Visibility of system status**: The device should keep the user informed about all the processes and state changes through feedback and in a reasonable time.

2. **Match between system and the real world**: The device should speak the users' language instead of system-oriented concepts and technicalities. The device should follow the real world conventions and display the information in a logical and natural order.

3. **User control and freedom**: The device should allow the user to undo and redo his actions, and provide clearly pointed "emergency exits" to leave unwanted states. These options should be preferably through a physical button or similar.

4. **Consistency and standards**: The device should follow the established conventions, on condition that the user should be able to do things in a familiar, standard and consistent way.

5. **Error prevention**: The device should hide or deactivate unavailable functionalities, warn users about critical actions and provide access to additional information.

6. **Minimize the user's memory load**: The device should offer visible objects, actions and options in order to prevent users to memorize information from one part of the dialogue to another.

7. **Customization and shortcuts**: The device should provide basic and advanced configuration options, allow definition and customization of (or to provide) shortcuts to frequent actions.

8. **Efficiency of use and performance**: The device should be able to load and display the required information in a reasonable time and minimize the required steps to perform a task. Animations and transitions should be displayed smoothly.

9. **Help users recognize, diagnose, and recover from errors**: The device should display error messages in a language familiar to the user, indicating the issue in a precise way and suggesting a constructive solution.

10. **Help and documentation**: The device should provide easy-to-find documentation and help, centered on the user's current task and indicating concrete steps to follow.

Heuristics for the Assessment of Interfaces of Mobile Devices

Olibário Machado Neto e Maria da Graça Pimentel

1. **Use of the screen space**: The interface should be designed so that the items are neither too distant, nor too stuck. Margin spaces may not be large in small screens to improve information visibility. The more related the components are, the closer they must appear on the screen. Interfaces must not be overwhelmed with a large number of items.

2. **Consistency and standards**: The application must maintain the components in the same place and look throughout the interaction, to facilitate learning and to stimulate the user's short-term memory. Similar functionalities must be performed by similar interactions.The metaphor of each component or feature must be unique throughout the application, to avoid misunderstanding.

3. **Visibility and easy access to all information**: All information must be visible and legible, both in portrait and in landscape. This also applies to media, which must be fully exhibited, unless the user opts to hide them. The elements on the screen must be adequately aligned and contrasted.

4. **Adequacy of the component to its functionality**: The user should know exactly which information to input in a component, without any ambiguities or doubts. Metaphors of features must be understood without difficulty.

5. **Adequacy of the message to the functionality and to the user**: The application must speak the user's language in a natural and non-invasive manner, so that the user does not feel under pressure. Instructions for performing the functionalities must be clear and objective.

6. **Error prevention and rapid recovery to the last stable state**: The system must be able to anticipate a situation that leads to an error by the user based on some activity already performed by the user. When an error occurs, the application should quickly warn the user and return to the last stable state of the application. In cases in which a return to the last stable state is difficult, the system must transfer the control to the user, so that he decides what to do or where to go.

7. **Ease of input**: The way the user provides the data can be based on assistive technologies, but the application should always display the input data with readability, so that the user has full control of the situation. The user should be able to provide the required data in a practical way.

8. **Ease of access to all functionalities**: The main features of the application must be easily found by the user, preferably in a single interaction. Most-frequently-used functionalities may be performed by using shortcuts or alternative interactions. No functionality should be hard to find in the application interface. All input components should be easily assimilated.

9. **Immediate and observable feedback**: Feedback must be easily identified and understood, so that the user is aware of the system status. Local refreshments on the screen must be preferred over global ones, because those ones maintain the status of the interaction. The interface must give the user the choice to hide messages that appear repeatedly. Long tasks must provide the user a way to do other tasks concurrently to the task being processed. The feedback must have good tone and be positive and may not be redundant or obvious.

10. **Help and documentation**: The application must have a help option where common problems and ways to solve them are specified. The issues considered in this option should be easy to find.

11. **Reduction of the user's memory load**: The user must not have to remember information from one screen to another to complete a task. The information of the interface must be clear and sufficient for the user to complete the current task.

Towards the Development of Usability Heuristics for Native Smartphone Mobile Applications

Ger Joyce e Mariana Lilley

1. **Provide immediate notification of application status**: Ensure the mobile application user is informed of the application status immediately and as long as is necessary. Where appropriate do this non-intrusively, such as displaying notifications within the status bar.

2. **Use a theme and consistent terms, as well as conventions and standards familiar to the user**: Use a theme for the mobile application to ensure different screens look alike. Also create a style guide from which words, phrases and concepts familiar to the user will be applied consistently throughout the interface, using a natural and logical order. Use platform conventions and standards that users have come to expect in a mobile application such as the same effects when gestures are used.

3. **Prevent errors where possible; Assist users should an error occur**: Ensure the mobile application is error-proofed as much as is possible. Should an error occur,

let the user know what the error is in a way they will understand, and offer advice in how they might fix the error or otherwise proceed.

4. **Display an overlay pointing out the main features when appropriate or requested**: An overlay pointing out the main features and how to interact with the application allows first-time users to get up-and-running quickly, after which they can explore the mobile application at their leisure. This overlay or a form of help system should also be displayed when requested.

5. **Each interface should focus on one task**: Being focusing on one task ensures that mobile interfaces are less cluttered and simple to the point of only having the absolute necessary elements onscreen to complete that task. This also allows the interface to be glanceable to users that are interrupted frequently.

6. **Design a visually pleasing interface**: Mobile interfaces that are attractive are far more memorable and are therefore used more often. Users are also more forgiving of attractive interfaces.

7. **Intuitive interfaces make for easier user journeys**: Mobile interfaces should be easy-to-learn whereby next steps are obvious. This allows users to more easily complete their tasks.

8. **Design a clear navigable path to task completion**: Users should be able to see right away how they can interact with the application and navigate their way to task completion.

9. **Allow configuration options and shortcuts**: Depending on the target user, the mobile application might allow configuration options and shortcuts to the most important information and frequent tasks, including the ability to configure according to contextual needs.

10. **Cater for diverse mobile environments**: Diverse environments consist of different types of context of use such as poor lighting conditions and high ambient noise are common ailments mobile users have to face every day. While the operating system should allow the user to change the interface brightness and sound settings, developers can assist users even more for example by allowing them to display larger buttons and allowing multimodal input and output options.

11. **Facilitate easier input**: Mobile devices are difficult to use from a content input perspective. Ensure users can input content more easily and accurately by, for instance displaying keyboard buttons that are as large as possible, as well as allowing multimodal input and by keeping form fields to a minimum.

12. **Use the camera, microphone and sensors when appropriate to lessen the users' workload**: Consider the use of the camera, microphone and sensors to lessen the users' workload. For instance, by using GPS so the user knows where they are and how to get there they need to go, or by using OCR and the camera to digitally capture the information the user needs to input, by allowing use of the microphone to input content which would save the user from having to type on the small keyboard.