Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Estudo Comparativo do Uso de React Native e Flutter para Aplicações de Realidade Aumentada

Victor Martins João

Monografia Final

mac 499— Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Alfredo Goldman
Cossupervisor: Renato Cordeiro Ferreira

São Paulo
24 de Dezembro de 2021

Agradecimentos

A Deus, por ter me concedido saúde e determinação para vencer todos os desafios enfrentados nesses anos de estudo. À minha esposa, pais e irmão, por me darem todo o suporte necessário para eu conseguir me dedicar aos estudos, principalmente durante o desenvolvimento deste trabalho. Aos meus colegas, por me auxiliarem no decorrer da graduação. Aos meus amigos, por tornarem mais leves os momentos de dificuldade. Aos professores, por compartilharem seus valiosos conhecimentos, que levo para a vida. Aos meus orientadores, por me instruírem com excelência e apresentarem tamanha paciência no período em que trabalhamos juntos. A todos aqueles que participaram do experimento de usabilidade, que enriqueceu os resultados dessa pesquisa.

Resumo

Victor Martins João. **Estudo Comparativo do Uso de React Native e Flutter para Aplicações de Realidade Aumentada**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Existem duas grandes vertentes de desenvolvimento móvel: o nativo, que permite maior controle e velocidade das aplicações desenvolvidas; e o híbrido, cujo diferencial é permitir a manutenção de apenas uma base de código. No entanto, quando são necessários recursos nativos, como é o caso da realidade aumentada, as abordagens híbridas precisam de adaptações. Esta pesquisa apresenta um estudo comparativo entre duas ferramentas de desenvolvimento móvel híbrido, React Native e Flutter, para realidade aumentada. Para isso, foi desenvolvida a aplicação Classic Blue usando o ARCore em cada ferramenta de desenvolvimento híbrido. A partir dessa aplicação foram analisados os seguintes aspectos: técnicos, como paradigmas de programação, manutenibilidade e adotabilidade das ferramentas; e de usabilidade, através de um experimento de usabilidade da aplicação, que avaliou tempo de conclusão das tarefas, dificuldade de conclusão e notas do questionário de usabilidade aplicado. Os resultados obtidos foram: para plataforma, o ARCore não consegue identificar planos verticais que não apresentem contraste; para ferramentas, o React Native apresentou vantagens sobre o Flutter nos aspectos técnicos e de usabilidade.

Palavras-chave: Desenvolvimento móvel híbrido. Realidade aumentada. Estudo comparativo. React Native. Flutter. Classic Blue.

Abstract

Victor Martins João. **Comparative Study of the Use of React Native and Flutter for Augmented Reality Applications**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

There are two main approaches to mobile development: native, which allows more control and speed of the developed applications; and hybrid, whose differential is to allow the maintenance of a single code base. However, when native resources are necessary like in augmented reality, the hybrid approach needs adaptations. This research presents a comparative study between two hybrid development tools, React Native and Flutter, for augmented reality. To that end, the Classic Blue application was developed using ARCore in each hybrid development tool. From this application, the following aspects were analyzed: technical, like programming paradigms, maintainability, and the tools adoptability; and usability aspects through a usability experiment of the application that evaluated completion time of the tasks, completion difficulty, and ratings of a usability questionnaire applied. The results were: for the platform, ARCore cannot identify vertical planes without contrast; for tools, React Native presented advantages over Flutter, both in technical and usability aspects.

Keywords: Hybrid mobile development. Augmented reality. Comparative study. React Native. Flutter. Classic Blue.

Lista de Abreviaturas

IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
API	Interface de Programação de Aplicação (*Application Programming Interface*)
MVP	Mínimo Produto Viável (*Minimum Viable Product*)
RACI	Responsável, Aprovador, Consultado, Informado (*Responsible, Accountable, Consulted e Informed*)
SWOT	Forças, Fraquezas, Oportunidades, Ameaças (*Strength, Weaknesses, Opportunities, Threats*)
HTML	Linguagem de Marcação de Hipertexto (*HyperText Markup Language*)
JSON	Notação de Objeto JavaScript (*JavaScript Object Notation*)
SLAM	Localização e Mapeamento Simultâneos (*Simultaneous Localization and Mapping*)
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos (*Institute of Electrical and Electronics Engineers*)
KF	Filtro de Kalman (*Kalman Filter*)
EKF	Filtro de Kalman Estendido (*Extended Kalman Filter*)
PF	Filtragem de Partículas (*Particle Filtering*)
VISLAM	Localização e Mapeamento Simultâneo Visual-Inercial (*Visual-Inertial Simultaneous Localization and Mapping*)
SIFT	Transformação de Característica Invariável em Escala (*Scale Invariant Feature Transform*)
DOG	Diferença de Gaussianas (*Difference of Gaussians*)
UEQ	Questionário de Experiência de Usuário (*User Experience Questionnaire*)
MADLC	Ciclo de Vida de Desenvolvimento de Aplicativos Móveis (*Mobile Application Development Lifecycle*)
APK	Pacote de Aplicativo Android (*Android Application Package*)

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introdução

O desenvolvimento móvel, por muitas vezes, visa o maior alcance de mercado possível. Para isso, os aplicativos devem ser compatíveis com os mais diversos modelos celulares e, consequentemente, com seus respectivos sistemas operacionais (em geral, Android ou IOS). Ao escolher uma ferramenta de desenvolvimento, duas vertentes entram nessa discussão: as ferramentas nativas e as híbridas.

Como propõem Vilček e Jakopec (2017), aplicações nativas são aquelas específicas para uma plataforma (Android ou IOS), e desenvolvidas utilizando linguagens próprias para ela. Já as aplicações híbridas são aquelas que podem ser utilizadas em diferentes plataformas, comportando-se como nativas, porém sendo desenvolvidas com outras tecnologias.

As tecnologias utilizadas para o desenvolvimento de aplicações híbridas, também chamadas ferramentas híbridas, se destacam no custo-benefício em relação ao orçamento e cronograma diante do número de plataformas que podem ser implementadas. Essa vantagem é criada pela possibilidade de reutilização de código entre diferentes plataformas. No entanto, códigos-fonte específicos de plataforma ainda podem ser necessários quando se trata de funcionalidades que usam recursos específicos (Meirelles *et al.*, 2019). Um exemplo é a realidade aumentada. Para implementá-la, é necessário usar alguma solução específica de plataforma, por exemplo, a API criada pelo Google chamada ARCore, que traz implementações nativas para esse tipo de tecnologia.

A realidade aumentada, como define, Kaufmann (2003), é a visualização do mundo real com objetos virtuais sobrepostos ou compostos nele. Esse tipo de tecnologia tem ganhado mais visibilidade no mercado. Em 2021, a Grand View Research, uma empresa indo-estadunidense de consultoria e pesquisa de mercado, publicou uma pesquisa sobre o tamanho do mercado de realidade aumentada. A pesquisa revelou que o mercado global de realidade aumentada foi avaliado em 2020 em 17,67 bilhões de dólares e com potencial taxa de crescimento anual de 43,8% até 2028. Esse aumento é esperado pela crescente demanda de assistência remota e também por empresas estarem explorando cada vez mais o potencial dessa tecnologia para oferecer uma experiência customizada e interativa para seus clientes (Research, 2021).

Dados os benefícios das ferramentas híbridas e a crescente demanda por realidade aumentada, este estudo propõe compreender os desafios e limitações da implementação

de aplicações de realidade aumentada, e assim auxiliar desenvolvedores a escolherem com mais propriedade qual ferramenta de desenvolvimento móvel híbrido é melhor para esse propósito. Para isso, foram escolhidas duas das ferramentas híbridas mais utilizadas no mercado segundo o Stack Overflow (Overflow, 2021), React Native e Flutter. Utilizando essas ferramentas, foi implementada uma mesma aplicação usando ARCore e feito um comparativo entre elas, tanto dos aspectos técnicos quanto de usabilidade.

A estrutura desse trabalho foi dividida da seguinte forma. O Capítulo 2 apresenta o protótipo da aplicação utilizada no estudo comparativo. O Capítulo 3 introduz um estudo das ferramentas de desenvolvimento híbrido, comparando seus aspectos técnicos. O Capítulo 4 lista os desafios e limitações da realidade aumentada. O Capítulo 5 descreve as implementações de realidade aumentada testadas durante o desenvolvimento. O Capítulo 6 relata o experimento de usabilidade realizado nesse estudo e seus resultados referentes a usabilidade das aplicações desenvolvidas. Por fim, o Capítulo 7 resume as conclusões tiradas a partir de todo o estudo desenvolvido.

Capítulo 2

Protótipo

Este capítulo visa apresentar o protótipo utilizado no estudo comparativo e introduzir o conceito de realidade aumentada aplicado a ele. É importante conhecer a aplicação para compreender os exemplos utilizados nos capítulos seguintes.

2.1 Classic Blue

Para atingir os objetivos do estudo comparativo, foi desenvolvimento o protótipo de uma aplicação *benchmark*, o aplicativo da Classic Blue. Ele foi idealizado para o projeto da disciplina Design de Interfaces do Departamento de Relações Públicas, Propaganda e Turismo da USP (CRP0558), ministrada pelo Prof. Dr. Luiz Guilherme de Carvalho Antunes. O projeto da disciplina consistia em idealizar uma aplicação móvel, desde o modelo de negócios da empresa, até o design final do aplicativo. Ele foi realizado pelos alunos, Victor Martins João (autor deste trabalho), Letícia Alves Lussietto, Pietro Augusto Gubel Portugal e Jhonatan Ferreira Alencar.

Classic Blue é uma empresa fictícia de assinatura de quadros decorativos. Com o objetivo de tornar mais acessível e rotativa a decoração de ambientes, a empresa ofereceria um serviço de rodízio de quadros selecionados por uma curadoria conforme as preferências do usuário. Para facilitar a interação com seus clientes, a empresa teria um aplicativo, que funcionaria como uma rede social exclusiva para os clientes, onde eles poderiam interagir com outros usuários, com artistas, e poderiam gerenciar suas assinaturas e comprar quadros.

2.2 Realidade Aumentada no Protótipo

O aplicativo da Classic Blue foi utilizado para esse estudo por sua funcionalidade de realidade aumentada. Uma definição proposta por Azuma, 1997, é que realidade aumentada é um sistema que possui três características principais: combinação de elementos reais e virtuais, interação em tempo real e reprodução de conteúdos ou elementos em 3D.

Em aplicações móveis, uma utilização comum de realidade aumentada é feita através da

câmera do dispositivo, em que elementos virtuais são exibidos na tela como se estivessem no mundo real. Um exemplo disso são os efeitos do aplicativo Instagram[1], que exibem elementos 3D sobre o rosto do usuário. Outro exemplo é o jogo Pokémon Go[2], que projeta o Pokémon na tela, como se estivesse em frente ao usuário, trazendo imersão ao universo do jogo.

De modo similar aos exemplos acima, a funcionalidade de realidade aumentada desenvolvida para o aplicativo Classic Blue também faz uso da câmera e exibe elementos 3D na tela do dispositivo. O Capítulo 5 apresenta as implementações desenvolvidas dessa funcionalidade, e a Subseção 2.4.3 descreve de forma detalhada a funcionalidade do aplicativo da Classic Blue.

2.3 Processo de Prototipação

Durante a disciplina, sob orientação do Prof. Dr. Luiz Guilherme, os estudantes realizaram tarefas para guiar o processo de prototipação. Inicialmente, foi necessário propor uma ideia de aplicativo simples, com apenas uma funcionalidade que gerasse valor para o usuário. A partir dessa ideia, foi elaborada uma proposta de valor, com o objetivo de explicar o que é o produto, para quem ele foi criado, onde e quando o produto deveria ser usado, e porque investir. Após a validação da proposta, foi descrito um MVP (*Minimum Viable Product*), seguindo o conceito proposto por Ries (2011). A próxima etapa nesse processo de descoberta foi o desenvolvimento de uma matriz RACI (*Responsible*, *Accountable*, *Consulted*, *Informed*) para definir os *stakeholders*, aqueles com influência sobre o projeto (Institute, 2013). Em seguida, aplicamos a matriz SWOT (*Strength*, *Weaknesses*, *Opportunities*, *Threats*), para avaliar as oportunidades e ameaças à ideia a partir das forças e fraquezas do modelo de negócios(Weihrich, 1982). Após essas informações, foi construído o *Business Model Canvas*, uma ferramenta que auxilia na definição do modelo de negócio de uma empresa (Barquet *et al.*, 2011). Por fim, as últimas etapas antes da prototipação de fato, foram: a elaboração de uma persona, que é a descrição de uma pessoa que representa o público alvo daquele produto; as pesquisas com usuários que se enquadram nesse público, para entender o problema da perspectiva deles; e o planejamento da aplicação com base nessas necessidades e sugestões identificadas.

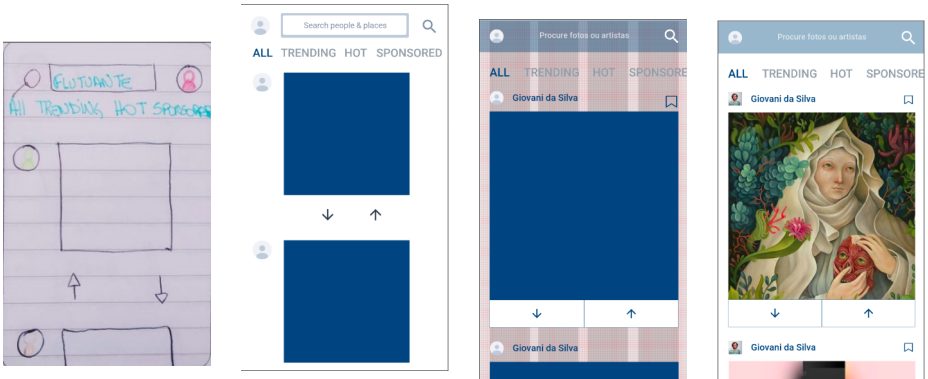
Os resultados desses estudos levaram ao desenvolvimento do protótipo em questão. No entanto, antes de chegar à sua versão final, o *design* passou por algumas etapas: rascunho, protótipo de baixa fidelidade, *grid*, e *layout* final. Esse processo de *design* pode ser consultado com mais detalhes no livro de Brown (2010), os exemplos de sua aplicação podem ser observados na Figura 2.1, e os resultados, na Figura 2.2.

Cada etapa do *design* foi uma pequena entrega proposta e validada pelo professor da disciplina, o que garantiu uma coerência no desenvolvimento ao longo de todo o processo. Toda a prototipação foi realizada utilizando uma plataforma específica para esse propósito chamada Marvel [3]. Ela permite desenhar telas e conectá-las, simulando a navegação de

[1] Link do Instagram: https://www.instagram.com/. Acesso em 28 Jul 2021

[2] Link do Pokémon Go: https://pokemongolive.com/pt_br/ Acesso em 28 Jul 2021

[3] Link do Marvel: https://marvelapp.com/. Acesso em 15 Ago 2021



(a) *Rascunho.* **(b)** *Protótipo de baixa fidelidade.* **(c)** Grid. **(d)** Layout *final.*

Figura 2.1: *Processo de* design *da tela de* FEED.

uma aplicação real.

O rascunho consiste em desenhar em papel o esperado da aplicação móvel, sem se preocupar com detalhes, apenas considerando as funcionalidades, a navegação e a interação do usuário. O protótipo de baixa fidelidade, em desenhar digitalmente os rascunhos, que permitiram uma percepção mais detalhada do produto. O grid, em aplicar regras de espaçamento e posicionamento para transformar o produto em mais agradável ao usuário, deixando-o mais próximo do objetivo a ser alcançado. Por último, o *layout final* consiste em apresentar o aplicativo com alta fidelidade. Para isso, ela trouxe elementos esperados no produto final, tais como cores, imagens, textos, e campos de digitação.

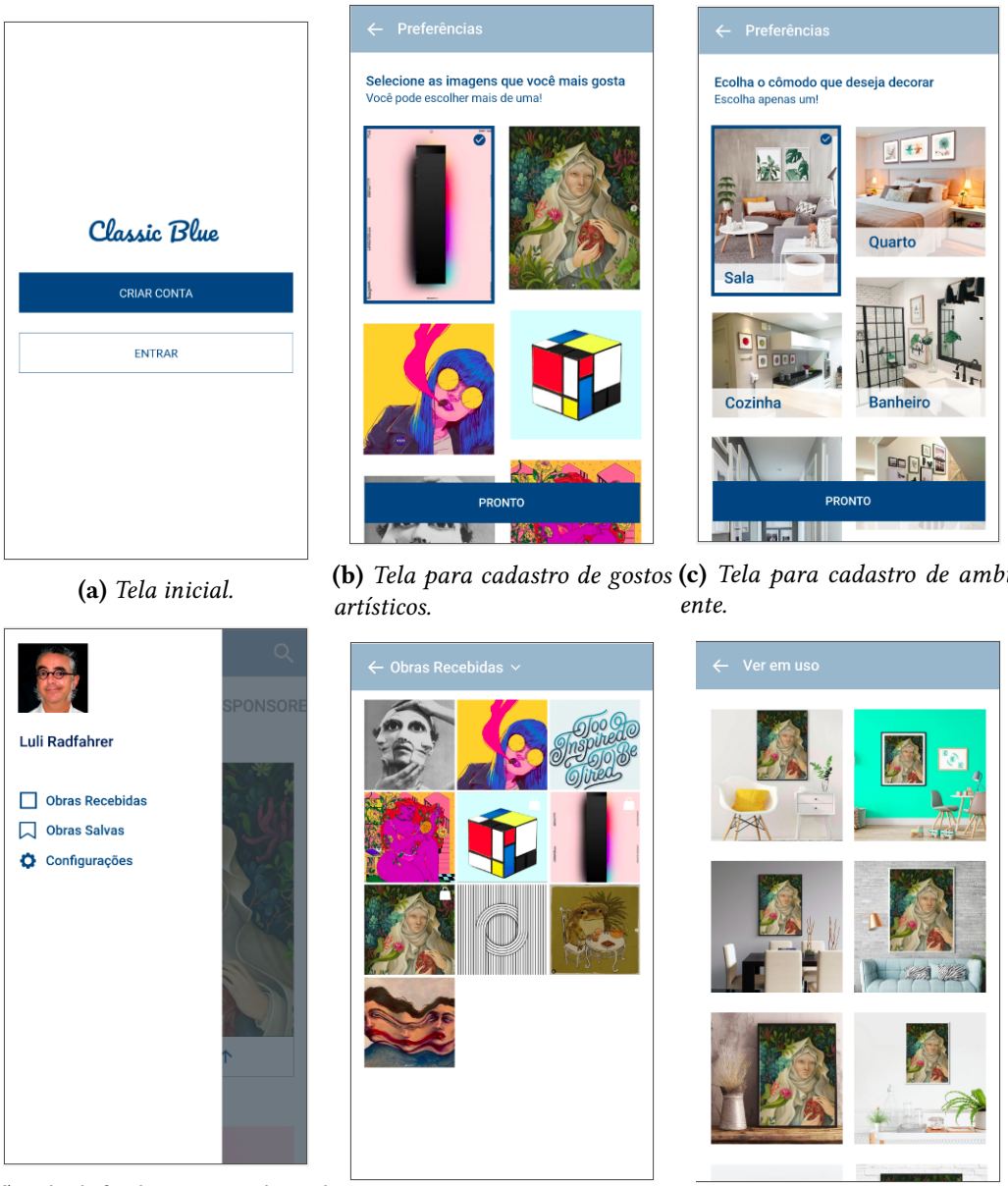
2.4 Funcionalidades

Durante a prototipação, uma série de funcionalidades foram planejadas, como por exemplo CADASTRO DE USUÁRIOS, AUTENTICAÇÃO, FEED, DETALHE DA IMAGEM, PERFIL DO USUÁRIO, PERFIL DO ARTISTA, COMPRA DE QUADROS, ASSINATURA E VER NA PAREDE. No entanto, foi selecionado um conjunto mínimo de funcionalidades que entregasse valor ao usuário e que implementasse a realidade aumentada, um dos critérios para utilização dessa aplicação neste estudo. Vale ressaltar que não foram considerados no desenvolvimento pontos como AUTENTICAÇÃO, CADASTRO DE USUÁRIOS, PERFIS, pois são funcionalidades básicas de um sistema e não agregariam no objetivo principal. Dessa maneira, as funcionalidades selecionadas para a implementação foram: FEED, DETALHE DA IMAGEM E VER NA PAREDE.

Antes de abordar sobre as ferramentas de desenvolvimento móvel híbridas no Capítulo 3, vamos olhar a seguir a descrição de cada uma das funcionalidades selecionadas.

2.4.1 Feed

O FEED é uma funcionalidade bem característica de redes sociais. No aplicativo Classic Blue, o FEED serve para a visualização das obras de artistas que o usuário demonstrou interesse durante o cadastro ou após ativamente segui-los. Essa tela foi projetada com



(a) *Tela inicial.*

(b) *Tela para cadastro de gostos artísticos.*

(c) *Tela para cadastro de ambiente.*

(d) *Tela do* feed *com menu lateral aberto.*

(e) *Tela de obras recebidas.*

(f) *Tela de ver as artes em uso.*

Figura 2.2: *Exemplo das telas do aplicativo após a etapa do* Layout *final.*

diversas interações como a avaliação de uma arte, salvamento da publicação, busca por texto, busca rápida, redirecionamento para perfil do artista e redirecionamento para o detalhe da arte.

Para o desenvolvimento da aplicação, como não foram implementados usuários, a funcionalidade do FEED consiste em uma listagem de imagens, buscadas via API do Unsplash [4], um famoso banco de imagens gratuitas para utilização. Nessa tela foi implementada a interação de redirecionamento para o detalhe da imagem.

A versão final do protótipo dessa tela é a apresentada na Figura 2.1d.

2.4.2 Detalhe da imagem

O DETALHE DA IMAGEM é uma tela onde é possível observar a arte de forma isolada e sem distrações, além de exibir a descrição dela segundo o autor. A página também possui algumas interações, tais como: VER NA PAREDE, Comprar e Ver fotos. Assim como no *feed*, nem todas as interações foram implementadas. O foco dessa funcionalidade foi garantir a interação VER NA PAREDE apenas.

O protótipo do detalhe da imagem pode ser observado na Figura 2.3

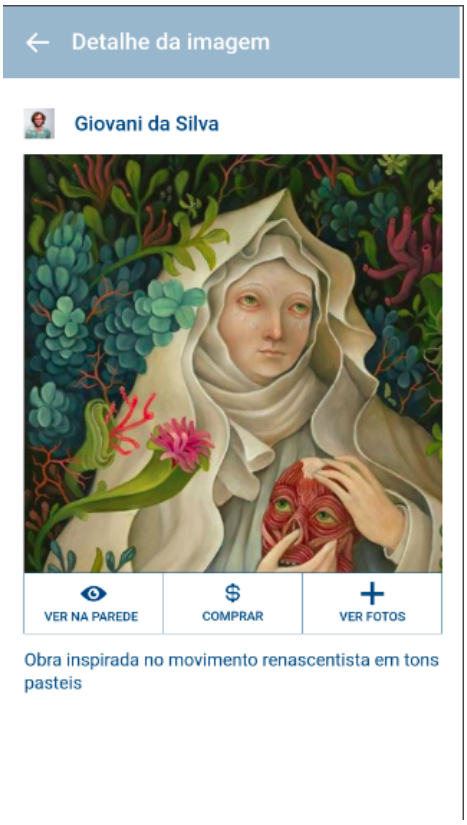


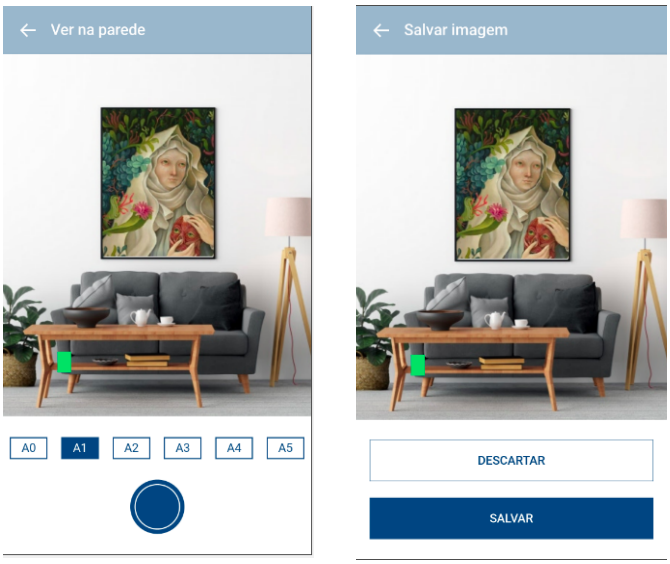
Figura 2.3: Layout *final da tela* DETALHE DA IMAGEM.

[4] Link para o site do Unsplash: https://unsplash.com/developers. Acesso em 23 Dez 2021

2.4.3 Ver na parede

A principal funcionalidade desse estudo é a VER NA PAREDE. Ela consiste em utilizar a tecnologia de realidade aumentada para projetar a arte selecionada no mundo real. Algumas interações foram planejadas para essa tela, como selecionar o tamanho da arte (A0, A1, A2, A3, A4 ou A5, dimensões comuns de papel), capturar a projeção 3D aplicada, bem como descartar a captura ou salvá-la na galeria do dispositivo.

Essa funcionalidade tem seu protótipo apresentado na Figura 2.4



(a) *Tela ver na parede.* **(b)** *Tela para salvar captura.*

Figura 2.4: *Telas da funcionalidade VER NA PAREDE.*

Capítulo 3

Ferramentas de Desenvolvimento Móvel Híbrido

Para este estudo, foram escolhidos duas ferramentas de desenvolvimento móvel híbrido, a fim de comparar qual das duas é melhor para aplicativos que usam realidade aumentada. As escolhidas foram React Native e Flutter. Para que a comparação fosse justa, foi realizado um estudo prévio dessas ferramentas e então desenvolvidas as aplicações utilizando as boas práticas aprendidas. Além disso, foi utilizada uma abordagem de desenvolvimento que se assemelhasse a um cenário comum no mercado de trabalho, ou seja, um cenário em que se tem um prazo curto e a necessidade de validar rapidamente uma ideia, levando o desenvolvedor a recorrer aos recursos já fornecidos pelas ferramentas ou pela comunidade.

Este capítulo, portanto, visa apresentar as ferramentas, compará-las em relação aos aspectos técnicos, e analisá-las nos quesitos manutenibilidade e adotabilidade. No Apêndice A são descritas também as ferramentas auxiliares utilizadas durante o desenvolvimento das aplicações.

3.1 Sobre as Ferramentas

React Native e Flutter são ferramentas de desenvolvimento móvel híbrido, isto é, com apenas um código-fonte é possível gerar código nativo para diferentes plataformas, sendo as principais Android e IOS. Elas são iniciativas *open-source* desenvolvidas por duas das maiores empresas de tecnologias do mundo, Facebook (React Native) e Google (Flutter). Essas ferramentas têm suas semelhanças e também suas diferenças, que serão abordadas a seguir.

3.2 Semelhanças

Pode-se dizer que as semelhanças entre elas são grandes visto que o Google abertamente declara que se inspirou no modelo de programação proposto pelo Facebook no React, que é uma ferramenta que utiliza JavaScript para desenvolvimento web sob o qual o React

Native foi construído (Flutter, 2021). Essas semelhanças ficam bem evidentes por meio dos principais paradigmas utilizados, como:

- misto de programação orientada a objetos com programação funcional, que é a possibilidade de utilizar classes e/ou funções de forma dinâmica;
- programação reativa, que é como as mudanças são propagadas pela a hierarquia do código; e
- composição, que é a forma de desenvolver baseada em união de pequenos elementos com responsabilidades específicas para gerar novos elementos mais complexos.

3.3 Diferenças

3.3.1 Linguagem

Apesar dos paradigmas em comum eles possuem algumas diferenças intrínsecas. A primeira delas são as linguagens utilizadas para o desenvolvimento. O React Native, como dito em sua documentação, é uma combinação das melhores partes do desenvolvimento nativo com React, que utiliza a linguagem JavaScript. Essa linguagem é bem difundida no desenvolvimento web, desde do início da própria web com os *scripts* embutidos no HTML, até a criação mais recente das ferramentas de desenvolvimento web como Angular, Vue e o próprio React. Ela foi criada para ser sintaticamente parecida com Java, porém com o propósito de ser mais dinâmica por meio da modificação de propriedades e comportamentos de objetos. Em sua essência, ela é linguagem sem tipagem estática, mas para alguns casos os tipos estáticos são muito úteis. Com isso em mente, a Microsoft desenvolveu o TypeScript, que adiciona funcionalidades ao JavaScript para que ela se torne uma linguagem com tipagem estática. O React Native permite que seja utilizado o próprio JavaScript ou que seja configurado para utilização do TypeScript, e para esse projeto foi escolhida a segunda abordagem, que garante maior integridade dos dados por todo do sistema. Já o Flutter foi construído utilizando uma linguagem própria, o Dart, que é uma linguagem otimizada para ser performática em qualquer plataforma. Ela foi desenhada para ser familiar, por isso acabou herdando muitas das características de outras linguagens como C, Java, JavaScript, entre outras. Algumas dessas características são a tipagem estática, chamadas assíncronas, geradores, *streams*, *getters* e *setters*.

3.3.2 Funcionamento

Ainda falando de suas diferenças, a forma como elas funcionam por baixo dos panos é bem única. O React Native funciona de uma forma relativamente simples. O conceito utilizado para transformar código em JavaScript em código nativo foi por meio da construção de uma "ponte", que utiliza mensagens em JSON para comunicar as mudanças de forma assíncrona, serializável e em lotes. Em outras palavras o código JavaScript é convertido em uma mensagem JSON, que por sua vez será convertida em código nativo. Essa abordagem é simples e poderosa, porém assim como em uma ponte, podem haver engarrafamentos, nesse de mensagens. O React Native tem trabalhado em formas de otimizar essa arquitetura,

inclusive em 2019 lançou o Hermes[1], que traz uma série de melhorias de desempenho para Android. O Flutter, diferentemente do React Native, não busca converter o código Dart em código nativo, ele busca utilizar a mesma abordagem de renderização utilizada pelas linguagens nativas, dessa forma ele acaba encurtando o caminho até o usuário, tendo consequentemente um desempenho melhor. Para isso ser possível, o Flutter define uma série de regras de hierarquia para que os elementos sejam renderizadas da forma correta na tela. Essa hierarquia desenvolvida por eles afeta como alguns limites são tratados e consequentemente altera a lógica de desenvolvimento em alguns casos, o que não ocorre no React Native.

3.4 Análise do Código

3.4.1 Exemplo de Código

Para evidenciar um pouco melhor essas semelhanças e diferenças, o Programa 3.1 e o Programa 3.2 apresentam, respectivamente, um trecho de código em React Native e outro em Flutter, que implementa a lista de categorias localizada na tela Feed do protótipo, como na Figura 3.1.

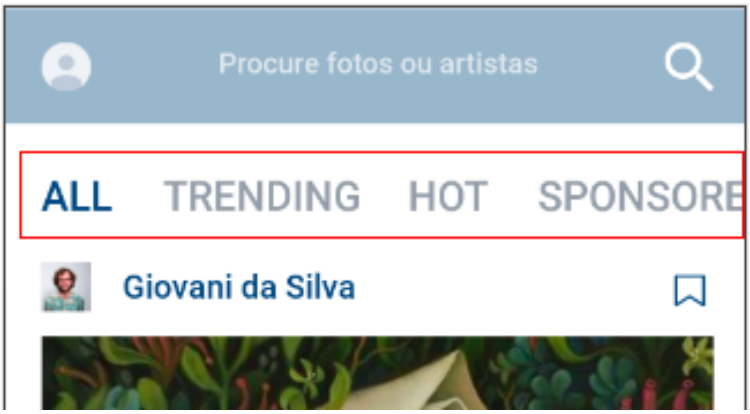


Figura 3.1: *Elemento de lista de categorias na tela de* feed *evidenciado pelo retângulo vermelho.*

3.4.2 Análise do Exemplo

A partir desses exemplos, vale ressaltar alguns pontos. O primeiro deles é que a composição fica bem evidente nos dois casos. Para fazer essa lista em React Native foram utilizados dois componentes, o FlatList que é o componente otimizado para fazer listas desenvolvido pelo próprio Facebook e o componente FeedCategory, um componente customizado desenvolvido durante esse estudo, ele é responsável por exibir uma categoria e implementar o comportamento clicável do texto. Em Flutter foram utilizados três Widgets, o Sized Box, responsável por definir uma altura para a lista, o ListView, para listar as categorias e o FeedCategory com o mesmo propósito do componente em React Native. Os dois primeiros widgets são da própria biblioteca do Flutter e o último é um widget customizado desenvolvido nesse estudo. Explorando um pouco mais os elementos,

[1] Link do Hermes: https://hermesengine.dev/. Acesso em 05 Set 2021

Programa 3.1 Componente FeedCategories no React Native.

```
1   const FeedCategories: React.FC = () => {
2     const feedCategories = ['ALL', 'TRENDING', 'HOT', 'SPONSORED'];
3     const [selectedCategory, setSelectedCategory] = useState(0);
4
5     return (
6       <FlatList
7         showsHorizontalScrollIndicator={false}
8         keyExtractor={(_, index) => `${index}`}
9         horizontal
10        contentContainerStyle={{
11          paddingHorizontal: spacing.horizontalSpace,
12        }}
13        data={feedCategories}
14        renderItem={({ item: category, index }) => {
15          const isSelectedCategory = selectedCategory === index;
16          return (
17            <FeedCategory
18              category={category}
19              isSelectedCategory={isSelectedCategory}
20              setSelectedCategory={() => {
21                setSelectedCategory(index);
22              }}
23            />
24          );
25        }}
26        ItemSeparatorComponent={({ }) => (
27          <View style={{ width: spacing.clickableSpace }} />
28        )}
29      />
30    );
31  };
```

observamos que a forma de utilizar difere em cada caso. O Flutter utiliza uma abordagem de chamada de função, recebendo as propriedades do Widget como argumentos dessa função. Esses argumentos podem ser de duas formas, nomeados ou posicionais. Os argumentos nomeados são aqueles que precisam ser especificados na chamada da função e não ficam restritos a ordem, já os argumentos posicionais ficam restritos a ordem definida pelo Widget. Ambos podem ser definidos como obrigatórios ou opcionais, mudando apenas a forma de declaração no Widget. Já em React Native a abordagem de utilização dos componentes é feita em *tags*, assim como no React, porém utilizando os componentes do React Native em vez das *tags* HTML. Por meio dessa abordagem a passagem das propriedades é feita como em HTML também, ficando a cargo do componente definir quais propriedades são obrigatórias e quais são opcionais. Além disso, em um projeto com TypeScript, é possível definir exatamente os tipos aceitos por cada propriedade.

Outro ponto interessante é o paradigma utilizado para a implementação dos elementos. O Flutter por padrão utiliza classes para a implementação dos Widgets, enquanto o React Native utiliza funções para a implementação dos componentes. Vale ressaltar que em React Native, utilizar funções não é um padrão, também é possível escrever utilizando classes.

Programa 3.2 Widget FeedCategories no Flutter.

```
1   class FeedCategories extends StatefulWidget {
2     @override
3     _FeedCategoriesState createState() => _FeedCategoriesState();
4   }
5
6   class _FeedCategoriesState extends State<FeedCategories> {
7     final List<String> _categories = [
8       'ALL',
9       'TRENDING',
10      'HOT',
11      'SPONSORED',
12    ];
13
14    int _selectedCategory = 0;
15
16    @override
17    Widget build(BuildContext context) {
18      return SizedBox(
19        height: 72,
20        child: ListView.builder(
21           padding: EdgeInsets.symmetric(horizontal: 4, vertical: 24),
22           scrollDirection: Axis.horizontal,
23           itemCount: _categories.length,
24           itemBuilder: (context, index) {
25             final String category = _categories[index];
26             final bool isSelectedCategory = _selectedCategory == index;
27             return FeedCategory(category, isSelectedCategory, () {
28               setState(() {
29                 _selectedCategory = index;
30               });
31             });
32           }),
33      );
34    }
35  }
```

Contudo, a partir da versão 16.8 do React, o Facebook introduziu o conceito de Hooks, a nova forma de usar estado sem utilização de classes. Ainda no tema estados, esse é mais um ponto em que as ferramentas se diferem. Os Hooks, utilizados no React Native, são funções utilizadas para conectar o estado do React e os ciclos de vida de um componente funcional. No exemplo acima vemos o Hook useState, uma função que recebe um estado inicial como argumento e devolve um vetor, que na primeira posição devolve o estado e na segunda posição uma função para alteração de estado. Essa abordagem apresentada pelo Facebook possui algumas vantagens como: a possibilidade de reutilizar lógicas baseadas em estados, de ter um código melhor segregado e independente do ciclo de vida do componente, e por fim, a possibilidade de implementar componentes puramente funcionais. Em Flutter, para utilização de estados é necessário utilizar um tipo de classe especial, a StatefulWidget. Essa classe exige a implementação do método createState, que por sua vez cria uma instância da classe do Widget propriamente dito, agora com estado. No Widget _FeedCategoriesState, o estado é uma variável mutável que é alterada pelo método setState da classe State, uma abordagem bem parecida com a do React com classes.

3.5 Discussão Sobre as Ferramentas

Na Seção 3.1 foram apresentadas as ferramentas, na Seção 3.2 e na Seção 3.3 suas semelhanças e diferenças. Esta seção, por sua vez, considera os aspectos técnicos descritos acima para comparar as ferramentas nos quesitos manutenibilidade e adotabilidade.

3.5.1 Manutenibilidade

De acordo com o *ISO/IEC 25010:2011* (2011) manutenibilidade é o grau de eficácia e eficiência com o qual um produto ou sistema pode ser modificado. Essas modificações incluem correções, melhorias, adaptações, bem como instalação de atualizações. Com isso em mente e com base no que foi observado durante o desenvolvimento das aplicações vamos discutir dois pontos que influenciam na manutenibilidade nessas ferramentas, que são: os paradigmas de programação utilizados e limitações nas atualizações de pacotes.

Na Subseção 3.4.2, foi apresentado que o Flutter utiliza predominantemente uma abordagem de classes, enquanto que o React Native se propõe a utilizar uma abordagem mais funcional. A utilização de classes pressupõe um acoplamento de estado muito forte, enquanto que o paradigma funcional permite uma liberdade maior, sendo assim mais fácil segregar código e consequentemente dar manutenção. Por esses motivos podemos dizer que o React Native tem uma vantagem em questão de manutenibilidade.

Considerando agora a manutenibilidade como a facilidade de instalação de atualizações, ambas podem apresentar problemas. Uma das grandes vantagens de se desenvolver utilizando ferramentas *open-source* é que a comunidade ou até mesmo empresas contribuem com as mais diversas bibliotecas, agilizando o desenvolvimento. No entanto, depender de algumas delas pode ocasionar problemas de manutenibilidade, como foi observado nesse estudo. Em React Native esse problema foi observado na biblioteca ViroReact, utilizada para o desenvolvimento de realidade aumentada. Essa biblioteca é bem poderosa, no entanto, ela se tornou *open-source* em 2019 e deixou de ser mantida pela empresa Viro Media, dessa forma ela só seria compatível com até a versão 0.59.9 do React Native e a versão 16.3.1 do

React, que não possui Hooks. Sendo assim, os desenvolvedores que dependessem dessa biblioteca, ou teriam que mudar a implementação de realidade aumentada para continuar acompanhando os avanços das outras bibliotecas, ou teriam que parar de atualizar as outras bibliotecas para continuar utilizando o ViroReact. No entanto, a comunidade de React Native assumiu a manutenção dessa biblioteca e criou o ViroCommunity para seguir dando as atualizações das bibliotecas as quais ela depende, impedindo que esse tipo de problema relatado acontecesse. Em Flutter ocorreu um problema semelhante. Em março de 2021, o Dart lançou a versão 2.12 que implementa o *Null Safety*, em outras palavras, variáveis não podem receber valor *null* a menos que seja declarado expressamente que podem. Apesar de ser uma atualização de adoção gradual, caso seja optado por fazê-la, gerará uma refatoração do código bem grande. Esse estudo foi iniciado em 2020, antes desse lançamento, e assim que lançado, foram realizadas algumas tentativas de migração para o *Null Safety*, porém a dependência de algumas bibliotecas da comunidade, como a biblioteca arcore_flutter_plugin[2] (em sua versão 0.0.10, na época do desenvolvimento), que não liberou uma versão oficial com *Null Safety*, impediram que essa migração fosse feita de forma completa. Esse problema demonstra uma comunidade pouco ativa e/ou imatura em relação a esse tipo de tecnologia (realidade aumentada).

Em geral, as duas ferramentas tem potencial para serem altamente manuteníveis dado o paradigma de composição, que permite uma boa segregação de responsabilidades e reutilização de código. Porém, o React Native apresenta maior manutenibilidade, através da utilização dos Hooks e também no suporte dado pela comunidade em relação à instalação e atualização de pacotes. O Flutter não atendeu as expectativas nesse quesito principalmente pela imaturidade da comunidade, que não fornece o suporte necessário para a manutenção de uma aplicação que contém realidade aumentada.

3.5.2 Adotabilidade

A pesquisa de Meyerovich e Rabkin, 2013 aponta conclusões bem interessantes relacionadas a adotabilidade de uma linguagem, a partir delas vamos discutir a respeito das linguagens utilizadas. A primeira conclusão é que linguagens de nicho, ou seja, usadas para contextos específicos, tendem a serem menos populares, enquanto que linguagens dominantes são mais populares até mesmo em nichos. O Dart, utilizado no Flutter pode ser considerada uma linguagem de nicho, visto que ela surge no contexto de desenvolvimento de aplicações móveis e se assemelha a linguagens desse nicho. Já o JavaScript, utilizado no React Native, é uma linguagem de ampla difusão, podendo ser utilizado para desenvolvimento web (HTML, React), desenvolvimento backend (Node.js) ou para desenvolvimento móvel como o próprio React Native. Por esse critério JavaScript é mais adotável do que Dart. No entanto, a estratégia de expansão de nichos do Flutter para contemplar também o desenvolvimento web, que ainda está em difusão, é muito importante para que a adotabilidade de Flutter seja maior com o passar do tempo.

Uma segunda conclusão dessa pesquisa é que linguagens com tipagem estática geram uma significante inquietação e desânimo. Para esse estudo, foi utilizada a abordagem com tipagem estática do JavaScript, o TypeScript, porém pode se desenvolver sem essa tipagem,

[2] Link da biblioteca: https://pub.dev/packages/arcore_flutter_plugin. Acesso em 23 Dez 2021.

enquanto que o Dart é uma linguagem com tipagem estática por natureza. Dessa forma, o JavaScript também leva vantagem em relação ao Dart no quesito adotabilidade.

Uma última conclusão desse estudo é que a escolha de uma linguagem está relacionada a bibliotecas, legado e familiaridade, sendo assim, existem fatores históricos atrelados a isso e consequentemente uma potencial estabilidade na adotabilidade das linguagens. Mais uma vez, identificamos que o JavaScript leva vantagem por ser uma linguagem que já existe a mais tempo do que Dart, assim como o React Native é mais antigo que Flutter. Como consequência podemos concluir que React Native possui uma comunidade de desenvolvimento mais madura, bem como possui mais bibliotecas auxiliares e por fim a linguagem utilizada é mais familiar.

Apesar de JavaScript ter vantagem de adotabilidade segundo os critérios acima, isso não significa que será sempre assim. A pesquisa também sugere que linguagens populares hoje podem ser substituídas ou aperfeiçoadas, e que essas mudanças podem ser duradouras. Por essa perspectiva, podemos dizer que o Dart, bem como o Flutter, tem possibilidade de superar o JavaScript e o React Native em relação a adotabilidade, uma vez que a linguagem amadureça e amplie o seu nicho, como no desenvolvimento web, por exemplo.

3.6 Conclusão Técnica

Comparando as ferramentas de desenvolvimento móvel híbridas nos aspectos técnicos observamos semelhanças em relação a conceitos como reatividade e composição, e diferenças em relação ao paradigma principal de desenvolvimento e estrutura de código. A lógica de desenvolvimento é bem semelhante nos dois casos, porém a identificação do React Native com outras ferramentas de desenvolvimento web facilitam o aprendizado inicial de quem tem essa expertise, enquanto que o Flutter se aproxima mais das ferramentas nativas, facilitando a adaptação nesse caso. Por fim, outro ponto destacado nesse estudo foi em relação à maturidade da comunidade e o quanto isso pode impactar na decisão das ferramentas, na qual a comunidade React Native demonstrou estar mais madura em relação a comunidade Flutter. Considerando os fatores manutenibilidade e adotabilidade, portanto, podemos concluir que o React Native é mais indicado para o desenvolvimento de aplicações que implementam realidade aumentada.

Capítulo 4

Realidade Aumentada

Como apresentada na Seção 2.2, a realidade aumentada é uma tecnologia que combina elementos reais com os virtuais. No caso dos dispositivos móveis, a partir da câmera e de outros sensores presentes, busca entender o ambiente ao seu redor para que a combinação seja o mais fiel possível de uma experiência esperada fisicamente, ou seja, caso o elemento virtual realmente estivesse ali.

Para implementar a realidade aumentada no protótipo em questão foi utilizada a plataforma do Google, específica para essa finalidade, chamada ARCore. Como diz em sua própria documentação (Google, 2021b), essa plataforma utiliza diversas APIs para permitir que o telefone perceba o ambiente, entenda o mundo e interaja com as informações. O ARCore utiliza três funcionalidades para combinar elementos virtuais com os elementos reais observados através da câmera:

- **Detecção de movimento**: Permite que o telefone mapeie sua posição relativa no mundo;
- **Entendimento do ambiente**: Permite que o telefone identifique tamanho e localização de superfícies (verticais e horizontais);
- **Estimativa de luminosidade**: Permite que o telefone estime as condições de luminosidade do ambiente;

O ARCore é uma plataforma que pode ser utilizada de várias formas. Pode ser utilizada diretamente em aplicações Android, IOS ou até mesmo em *game engines* como a Unity e a Unreal. Nesse projeto as implementações foram testadas apenas em dispositivos Android, essencialmente por não ter acesso a dispositivos IOS durante o desenvolvimento, então fica para um trabalho futuro testar em IOS e verificar se esse comparativo também é válido.

A seguir vamos olhar mais a fundo sobre os elementos que tornam a funcionalidade possível e quais são os desafios da realidade aumentada, principalmente tratando de dispositivos móveis.

4.1 SLAM

Um dos pontos cruciais dessa tecnologia é a detecção de movimento. Para isso o ARCore utiliza um processo chamado SLAM (*Simultaneous Localization and Mapping*), que permite entender onde o telefone está em relação ao mundo ao seu redor (Google, 2021c). Como retratam Durrant-Whyte e Bailey (2006) em seu artigo, o SLAM começou a ser endereçado como uma solução para o mapeamento e localização na robótica em 1986, na conferência de Robôs e Automação da IEEE. Naquela época, métodos probabilísticos estavam começando a ser inseridos na comunidade sobre robótica. Uma série de métodos teóricos de estimativa eram aplicados pelos pesquisadores a fim de solucionar esse problema. Fora reconhecido nessa conferência que um mapeamento probabilístico consistente era um problema fundamental na robótica com grandes limitações conceituais e computacionais ainda. Somente em 1995 o SLAM foi formulado como um processo em que um robô consegue mapear o ambiente enquanto utiliza esse mapa para deduzir sua localização sem necessidade de um conhecimento prévio. Esse robô utiliza sensores para identificar as marcações definidas no ambiente e a partir da sua movimentação e de novas observações consegue inferir sua localização.

As múltiplas observações de uma marcação durante a movimentação permitem atualizar a localização estimada do robô e da marcação, bem como permitem propagar essa atualização para as outras marcações, mesmo que não sejam observadas a partir da nova localização. Isso ocorre por as marcações serem fortemente correlacionadas (suas localizações relativas são bem conhecidas) a partir das observações anteriores. A medida que novas marcações são descobertas, elas são imediatamente ligadas ou correlacionadas ao resto do mapa, dessa forma todas juntas acabam formando uma rede ligada por suas localizações relativas. Com isso, podemos concluir que a precisão da localização do robô relativa ao mapa é limitada apenas pela qualidade do mapa e pelas medições do sensor. Na figura 4.1 podemos ver a trajetória do robô ao longo do tempo, bem como as marcações e suas localizações estimadas, calculadas conforme as seguintes variáveis:

- x_k: Representa o vetor de localização e orientação do robô no instante k;
- u_k: Vetor de deslocamento para o estado x_k em relação ao instante anterior $k-1$;
- m_i: Vetor da localização da i-ésima marcação;
- z_{ik}: Observação da i-ésima marcação no instante de tempo k;

A partir dessas informações é possível calcular a distribuição probabilística para cada instante k. Nesse caso usamos uma distribuição posteriori conjunta para estimar a posição do robô (x_k), bem como a posição das marcações (m), dadas observações das marcações ($Z_{0:k}$), os descolamentos realizados até o momento ($U_{0:k}$), e a posição inicial do robô (x_0), como a probabilidade a seguir

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \tag{4.1}$$

Na prática, esse cálculo é feito de forma recursiva a partir da probabilidade calculada no instante $k-1$ utilizando o Teorema de Bayes. Para isso são necessários dois modelos,

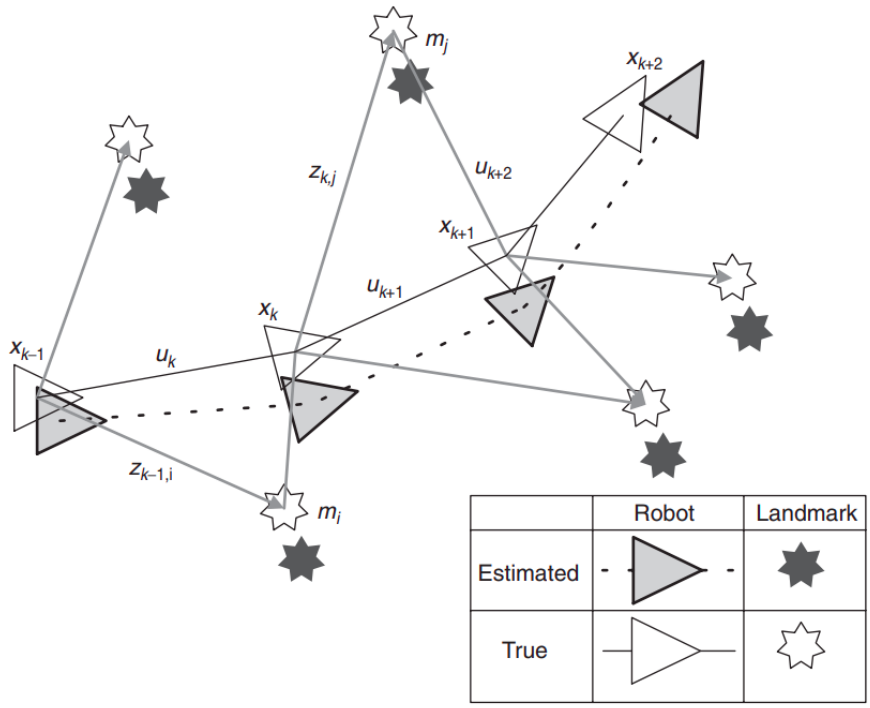


Figura 4.1: *Exemplo da trajetória do robô ao longo dos instantes $k-1$ a $k+2$ observando as marcações m (Durrant-Whyte e Bailey, 2006).*

um para movimentações e outro para as observações, representados da seguinte forma respectivamente:

$$P(x_k|x_{k-1}, u_k) \tag{4.2}$$

$$P(z_k|x_k, m) \tag{4.3}$$

Com isso o algoritmo do SLAM pode ser representado como a seguinte recursão de duas etapas

$$P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k|x_{k-1}, u_k) \times P(x_{k-1,m|Z_{0:k-1},U_{0:k-1},x_0})\, dx_{k-1} \tag{4.4}$$

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|x_k, m)P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})} \tag{4.5}$$

Tendo em mente a estrutura do problema, as soluções buscam encontrar uma representação apropriada para os modelos de movimentação e observação para otimizar a recursão acima, sendo os algoritmos EKF-SLAM e FastSLAM dois dos mais importantes.

4.1.1 EKF-SLAM

O EKF-SLAM é um algoritmo que aplica o *Extended Kalman Filter* (EKF) para estimar um valor para os modelos de movimentação e observação do SLAM. Para entender melhor o algoritmo, vale explicar de forma breve os conceitos por trás dele. O *Kalman Filter* (KF) é um sistema linear, discreto e com variação de tempo finita que calcula uma estimativa para um estado de forma a minimizar o erro quadrático médio (M. Ribeiro e I. Ribeiro, 2004). Em uma abordagem mais prática, o KF consiste em três equações, cada uma envolvendo manipulação de matrizes (Simon, 2001), que são:

$$K_k = AP_kC^T(CP_kC^T + S_z)^{-1} \tag{4.6}$$

$$\hat{x}_{k+1} = (A\hat{x}_k + Bu_k) + K_k(Y_{k+1} - C\hat{x}_k) \tag{4.7}$$

$$P_{k+1} = AP_kA^T + S_w - AP_kC^TS_z^{-1}CP_kA^T \tag{4.8}$$

Dessa forma, podemos observar que a estimativa de um estado no instante $k+1$ é calculada recursivamente de acordo com alguns valores no instante k. As variáveis para esse cálculo são as seguintes:

- K_k: Ganho de Kalman, que minimiza a covariância do erro;
- A, B, C: São matrizes que fazem parte das equações de entrada ($x_{k+1} = Ax_k + Bu_k + w_k$) e saída ($y_k = Cx_k + z_k$);
- w_k, z_k: São ruídos de processo e de medição, respectivamente;
- S_w, S_z: São as matrizes de covariância dos ruídos de processo e medição, respectivamente;
- P_k: Estimativa da matriz de covariância do erro no instante k. Note que ela também é calculada de forma recursiva, de acordo com o valor da estimativa anterior;

Compreendendo o KF, fica mais fácil entender o EKF, que nada mais é que a utilização do KF para sistemas com funções não-lineares. Esse é o caso do SLAM, na qual os modelos de movimentação e observação podem descritos, respectivamente, como

$$P(x_k|x_{k-1}, u_k) \iff x_k = f(x_{k-1}, u_k) + w_k \tag{4.9}$$

$$P(z_k|x_k, m) \iff z_k = h(x_k, m) + v_k \tag{4.10}$$

Nessa nova representação, as funções f e h são não-lineares e w_k e v_k são os ruídos com distribuição normal. Como KF assume uma distribuição Gaussiana, para que suas propriedades sejam mantidas é necessário realizar um processo de linearização dessas funções em relação à estimativa atual (M. Ribeiro e I. Ribeiro, 2004).

Pensando em complexidade computacional, essa abordagem pode não ser tão eficiente, pois a cada nova observação feita é necessário recalcular a matriz de covariância P, significando que a computação cresce de forma quadrática em relação ao número de marcações (Durrant-Whyte e Bailey, 2006), dificultando o processamento em tempo real para ambientes maiores.

4.1.2 FastSLAM

O FastSLAM é um algoritmo que traz uma reformulação para o problema de SLAM. Ele parte da constatação de que conhecer o caminho do robô torna as medições das marcações independentes, dessa forma, determinar as marcações pode ser desacoplado em diversos problemas de estimação diferentes, uma para cada marcação (Montemerlo *et al.*, 2002). Dada essa independência condicional, podemos reformular a probabilidade original substituindo a posição do robô no instante k (x_k) por $X_{0:k}$, que representa a trajetória do robô até o instante k. Isso implica que a posteriori pode ser calculada, portanto, da seguinte forma:

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0) \prod_k P(m_k|X_{0:k}, Z_{0:k}) \tag{4.11}$$

A partir dessa reformulação, o FastSLAM aplica o conceito de Rao-Blackwellization (R-B), que diz que um estado conjunto pode ser particionado de acordo com a regra do produto $P(x_1, x_2) = P(x_2|x_1)P(x_1)$ e se $P(x_2|x_1)$ puder ser representado de forma analítica, então apenas $P(x_1)$ precisa ser amostrado $x_1^{(i)} \sim P(x_1)$ (Durrant-Whyte e Bailey, 2006). Considerando esse contexto, podemos aplicar *Particle Filtering* (PF) para calcular a posteriori da trajetoria $P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0)$ e aplicar KF para estimar a posição das marcações $P(m_k|X_{0:k}, Z_{0:k})$. Essa abordagem é bem interessante, pois diminui a quantidade de amostras necessárias para se obter uma boa aproximação, o grande problema do PF.

Para entender melhor o algoritmo, vamos olhar o que é o PF e como ele e o KF são aplicados. Começando pelo PF, a ideia dele é representar uma posteriori por um conjunto de amostras retirados dessa posteriori. Essa representação é aproximada, mas é não-parametrizada, então fica livre para ser aplicada a mais distribuições além da Gaussiana (Thrun *et al.*, 2005), que o KF assume, por exemplo. Na prática, ele pode ser descrito pelas seguintes etapas, executadas de forma recursiva, que são:

- **Amostragem**: Nessa etapa, para cada uma das M amostras, é estimado um novo estado com base no anterior e no vetor de deslocamento. Essa estimativa é feita a partir da distribuição de transição de estado;
- **Fator de importância**: É calculado para cada amostra a razão entre a distribuição desejada e a distribuição proposta. Essa razão ajuda a ponderar a diferença entre as distribuições, como um fator de correção entre elas;
- **Reamostragem**: Esse processo basicamente faz M retiradas com reposição do conjunto atual, na qual a probabilidade de uma amostra ser retirada é proporcional ao peso dela. Cada amostra retirada é adicionada em um novo conjunto de amostras;

A aplicação do PF ao problema do SLAM consiste na utilização do modelo probabilístico

da movimentação para gerar um conjunto de amostras que representam as hipóteses da trajetória e assim realizar as atualizações conforme o processo descrito acima.

Como o FastSLAM utiliza essa abordagem de amostragem das trajetórias e como a estimativa das marcações dependem delas, o KF está associado individualmente a uma amostra. Portanto, para M amostras e K marcações, isso vai resultar num total de KM *Kalman Filters*, cada um de dimensão 2, devido às duas coordenadas das marcações (Montemerlo *et al.*, 2002). A grande vantagem desse algoritmo em relação ao EKF-SLAM é justamente o ganho de desempenho, que passa a ser $O(KM)$ em vez de ser $O(K^2)$, por utilizar pequenas matrizes de covariância P em vez de uma única matriz de dimensão $K \times K$.

4.2 VISLAM

Vimos que uma parte importante do SLAM é a identificação de uma marcação para medir sua posição em relação ao robô. Quando falamos de realidade aumentada, não estamos falando de robôs avaliando marcações, estamos falando do ser humano interagindo com um dispositivo para gerar a combinação do real com o virtual. Para isso o *visual-inertial* SLAM (VISLAM) utiliza informações visuais combinadas com medições inerciais de giroscópios ou acelerômetros, por exemplo, para estimar a posição e orientação da câmera em relação ao mundo. Essa é a abordagem utilizada pelo ARCore (Google, 2021c), que se aproveita do fato dos novos celulares hoje serem equipados com câmeras de qualidade cada vez melhores e também dispositivos de medição inercial para fazer com que essa funcionalidade seja cada vez mais viável.

De acordo com o estudo apresentado por Liu *et al.* (2018), existem dois grandes desafios computacionais para o VISLAM: O desafio do *frontend* e do solucionador. O *frontend* inclui as tarefas de extração e correlação de pontos importantes observados. Essa parte pode ser paralelizável e realizada de forma eficiente, em geral. Já a tarefa do solucionador é combinar as informações visuais e medições inerciais para otimizar as funções objetivo do SLAM apresentadas anteriormente (função de movimentação e observação). O VISLAM não necessita de muitos pontos importantes para atingir uma precisão razoável, uma vez que as medições inerciais fornecem restrições adicionais, porém a depender da solução utilizada e da implementação essa tarefa pode ser de grande custo computacional, como foi discutido anteriormente.

As tarefas do solucionador, em geral, são o gargalo para esse tipo de abordagem, uma vez que elas tem complexidade computacional maior em relação as de *frontend*. Contudo, problemas na identificação dos pontos importantes podem fazer com que essa segunda tarefa seja o maior dificultador desse algoritmo.

Exitem vários métodos para a identificação de pontos importantes, vamos abordar dois métodos importantes nessa área, o *Harris Corner Detector* e o *Scale Invariant Feature Transform*(SIFT).

4.2.1 Harris Corner Detector

Segundo a análise feita por Sánchez *et al.* (2018), o método *Harris Corner Detector*, apesar do surgimento de novas técnicas, continua sendo uma referência. Ainda em sua pesquisa, eles definem esse método como uma técnica para localizar pontos importantes em uma imagem a partir de uma função de autocorrelação, usada para medir a intensidade das diferenças encontradas em uma janela dessa imagem aplicando diversos deslocamentos.

De forma prática, esse método utiliza soma do quadrado das diferenças e busca encontrar cantos através de uma janela de pontos x que maximizam essa função para pequenos deslocamentos h:

$$E(h) = \sum_x w(x)(I(x+h) - I(x))^2 \tag{4.12}$$

Na função acima, $I(x)$ representa uma função de intensidade da imagem e $w(x)$ descreve a região de aceitação, em geral, definido por uma função Gaussiana. Aplicando a expansão da série de Taylor, podemos aproximar $I(x+h) \simeq I(x) + \nabla I(x)^T h$. Substituindo essa aproximação na função original e simplificando ela, chegamos na seguinte aproximação:

$$E(h) \simeq \sum_x w(x)(h^T \nabla I(x) \nabla I(x)^T h) = h^T \left(\sum_x w(x) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) h \tag{4.13}$$

Analisando a forma final dessa aproximação podemos concluir que o valor máximo pode ser encontrado analisando a seguinte matriz M, chamada matriz de autocorrelação:

$$M = \sum_x w(x) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{4.14}$$

A partir dessa matriz M, Harris e Stephens (1988) propuseram uma medida de resposta R para a detecção de cantos, que dispensa a utilização de seus autovalores λ_1 e λ_2 de forma explicita. Portanto, a partir do $Det(M) = \lambda_1 \lambda_2 = I_x^2 I_y^2 - (I_x I_y)^2$, $Tr(M) = \lambda_1 + \lambda_2 = I_x^2 + I_y^2$ e de uma constante k (geralmente entre 0,04 e 0,06), o valor de R pode ser calculado:

$$R = Det(M) - kTr(M)^2 \tag{4.15}$$

O valor de R pode ser traduzido nos seguintes resultados:

- R **é negativo**: Significa que a diferença entre os autovalores é grande e o *pixel* provavelmente pertence a uma aresta;
- R **é positivo e alto**: Os dois autovalores são altos e provavelmente é um canto;
- R **é positivo e baixo**: Os dois autovalores são baixos e faz parte de uma região plana;

Esse método é eficaz para o mapeamento de cantos, arestas e regiões planas e pode ser resolvido para otimizar o custo computacional como propõe Han *et al.* (2018). No entanto, a falta de cantos e arestas pode dificultar uma real compreensão da imagem.

4.2.2 SIFT

O segundo método para identificação de pontos importantes que vamos abordar é o *Scale Invariant Feature Transform* (SIFT). Esse método foi proposto por David G. Lowe e tem como objetivo transformar as informações de uma imagem em coordenadas resistentes a escala relativas a pontos locais (Lowe, 2004). Essa abordagem busca descrever pontos importantes de forma única, para ser possível correlacionar um ponto de forma eficiente quando comparado a pontos previamente calculados de um banco de imagens. Para chegarmos nessa descrição são necessárias quatro etapas: Detecção de extremos em espaço de escala, Localização de pontos chaves, Atribuição de Orientação e Descritor de pontos-chave.

Na primeira etapa de detecção de extremos em espaço de escala, são identificados candidatos a pontos-chave através de uma função de Diferença de Gaussianas (DOG). A ideia principal dessa etapa é criar diversas cópias da imagem e aplicar de forma incremental a convolução de uma função Gaussiana com a imagem, a partir de um fator de escala σ. Portanto, definimos a função L como sendo a imagem suavizada pela função Gaussiana G e D é a diferença de duas Gaussianas separadas por uma constante k, como representado abaixo:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{4.16}$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{4.17}$$

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{4.18}$$

Para calcular D de forma eficiente, são utilizadas conjuntos de imagens chamados de oitavas, na qual, cada oitava deve conter um número $s + 3$ de imagens, sendo s o número de incrementos. Considerando esse número de incrementos, o valor do incremento entre cada imagem pode ser definido em função de s na forma $k = 2^{1/s}$. Uma vez finalizado o cálculo de D em uma oitava, o processo é repetido para a próxima oitava, porém com um número de amostras da imagem 2 vezes menor.

A partir dos valores calculados de D, definimos que um ponto é de extremo em espaço de escala se ele tem valor maior ou menor que todos os seus vizinhos, tanto os oito vizinhos de sua imagem atual, quanto os nove da escala anterior e os outros nove da posterior.

A segunda etapa de localização de pontos-chave consiste em eliminar candidatos apontados na etapa anterior que possuam baixo contraste ou que pertençam a arestas, de forma parecida com o que propõe o *Harris Corner Detector*. Para eliminar pontos com baixo

contraste é aplicada a expansão da série de Taylor para D e calculado o valor aproximado da função para aquele extremo $\hat{x}$ da seguinte forma:

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{4.19}$$

Se o valor absoluto de $D(\hat{x})$ for menor que um determinado valor, então ele será descartado. Já para eliminar pontos que pertencem a arestas, assim como no *Harris Corner Detector*, podemos calcular $Det(H)$ e $Tr(H)$ sem conhecimento explicito dos autovalores de H, a matriz Hessiana de D. Considerando r, a razão entre os autovalores λ_1 e λ_2, para $\lambda_1 > \lambda_2$, podemos dizer que:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1\lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2^2} = \frac{(r+1)^2}{r} \tag{4.20}$$

Logo para decidirmos se vamos eliminar um candidato a ponto-chave, basta calcular se $Tr(H)^2/Det(H)$ é menor que $(r+1)^2/r$, para um determinado valor de r.

Na etapa de atribuição de orientação, como o próprio nome diz, buscamos atribuir uma orientação para os pontos chaves filtrados na segunda etapa. A partir das escalas dos pontos chaves são selecionadas as imagens suavizadas L e computados dois valores para cada ponto dessa imagem, a orientação θ e a magnitude m, através da diferença entre *pixels* adjacentes:

$$\theta(x,y) = tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))) \tag{4.21}$$

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{4.22}$$

Com esses valores é feito um histograma das orientações ao redor de um ponto-chave. Esse histograma é separado em 36 intervalos de 10 graus, e o valor adicionado ao histograma é ponderado pela magnitude naquele ponto e também por uma janela circular de peso Gaussiano, com σ 1,5 vezes maior do que o daquela escala, assim pontos mais próximos tem maior relevância do que pontos mais distantes. Após a criação desse histograma, a orientação com maior pico é atribuída ao ponto-chave em questão, além disso, caso existam picos de magnitude com pelo menos 80% em relação ao maior, são criados outros pontos-chave com a mesma localização e escala, porém com essas outras orientações.

Por fim, a etapa descritor de pontos-chave propõe uma descrição mais ampla do ponto-chave, considerando não só sua localização, escala e orientação, mas também as características da região ao seu redor, aumentando assim a robustez do método. Para isso, a abordagem utilizada é parecida com a etapa anterior, também observa as magnitudes e orientações dos pontos próximos ao ponto-chave dentro de uma janela circular definida por uma função Gaussiana. Contudo, em vez de criar um histograma para o ponto-chave, são criados histogramas de oito intervalos de 45 graus, para cada região de tamanho

4x4 pertencente a essa janela. As entradas desse histograma são também como na etapa anterior, porém considerando apenas os pontos da região na qual ele representa. Dessa forma, em uma janela de tamanho 16x16, são criados 4x4 histogramas contendo, cada um, 8 orientações com suas magnitudes associadas. Portanto, o descritor de um ponto-chave nada mais é que um vetor, que para cada região resultante desse processo, armazena as 8 orientações com suas magnitudes associadas.

Como mencionado anteriormente, essas etapas são realizadas com o propósito de tornar eficiente a correlação entre os pontos chaves encontrados na imagem com pontos-chave calculados a partir de um banco de imagens. Com isso, ao final dessas etapas, essa correlação é realizada para cada ponto-chave de forma independente e o melhor candidato para a correlação é definido pelo cálculo da distância Euclidiana mínima entre os vetores descritores.

De forma prática, esse método é bom para detectar pontos-chave e também para reconhecimento de objetos em uma imagem, mesmo com variações de rotação, escala e até mesmo oclusão. Essas características favorecem a utilização desse método para atacar o problema do SLAM, como apresentam Se *et al.* (2001) em sua pesquisa.

Após analisar os dois métodos de reconhecimento de pontos importantes apresentados, podemos observar que pontos de baixo contrastes são descartados. Essa limitação é comum para esse tipo de abordagem, baseada em pontos importantes, como é o caso do ARCore. No estudo feito por (Nowacki e Woda, 2020), são realizados uma série de experimentos para testar as capacidades do ARCore e do ARKit, a biblioteca de realidade aumentada da Apple. Em um desses testes, foi analisada a quantidade de pontos importantes encontrados em diferentes superfícies e constatado que numa parede de cor única, o ARCore não conseguiu encontrar pontos suficientes para mapear a superfície vertical e o ARKit conseguiu mapear de forma parcial e demorou cerca de 40 segundos, dificultando a utilização em tempo real. Sendo assim, fica inviável a utilização do ARCore para a identificação de planos verticais em paredes de cor única, um caso bem comum e fundamental para a funcionalidade de realidade aumentada do *Classic Blue*.

4.3 Aprendizados

Nas seções anteriores 4.1 e 4.2, vimos que o SLAM é, em geral, um gargalo para a realidade aumentada devido a sua maior complexidade computacional, porém a abordagem baseada em identificação de pontos importantes é ruim para imagens com baixo contraste, que é o caso de uma parede de cor única, fazendo com que essa etapa, chamada de *frontend* seja o real gargalo. Essa limitação faz com que o ARCore não seja indicado para o reconhecimento de planos verticais com baixo contraste, levando os desenvolvedores e *stakeholders* a buscarem alternativas como serão apresentadas no próximo capítulo.

Capítulo 5

Implementando Realidade Aumentada

Neste capítulo serão apresentadas as implementações de realidade aumentada propostas durante o desenvolvimento dessa pesquisa, considerando as limitações apresentadas no Capítulo 4. As três implementações testadas foram: planos verticais no ARCore, planos verticais na Unity e reconhecimento de imagem.

5.1 Implementações Realizadas

5.1.1 Planos verticais no ARCore

A primeira solução testada foi a utilização da API do ARCore para reconhecer planos verticais e consequentemente posicionar o quadro virtualmente, de forma a simular o esperado no mundo real.

Em React Native a API do ARCore foi utilizada através de uma biblioteca chamada ViroReact. Como a API do ARCore está disponível de forma nativa, ou seja, não híbrida, seria necessário implementar a integração do React Native com o código nativo manualmente. Essa biblioteca faz exatamente esse trabalho, poupando tempo de desenvolvimento. Como relatado anteriormente na seção 3.5.1, a integração com essa biblioteca foi dificultada pelo fato da empresa ter descontinuado esse projeto e não ter documentado de forma clara. Então inicialmente, a integração dependia de recursos mais antigos, tornando o processo bem desafiador. Só foi possível completar a integração após encontrar uma atualização da comunidade, que resolveu todas as incompatibilidades encontradas anteriormente.

Uma vez finalizada a integração e essa primeira implementação da funcionalidade, foi constatado exatamente a limitação teórica apresentada. Não foram identificados planos verticais em uma parede de cor única e sem textura, impossibilitando assim o posicionamento dos quadros, como propõe a funcionalidade.

Para o Flutter, também seria necessária uma adaptação manual da API nativa do ARCore, no entanto, foi utilizada uma integração já implementada pela comunidade

chamada arcore_flutter_plugin [1] . Diferentemente do React Native a integração foi bem fluida e a implementação ocorreu sem imprevistos. Porém, a limitação foi a mesma citada acima, não foram identificados planos verticais, impossibilitando essa solução.

5.1.2 Planos verticais na Unity

A segunda solução testada também busca identificar planos verticais, porém dessa vez utilizando a implementação do ARCore para a *game engine* Unity. A ideia era descartar a possibilidade das bibliotecas testadas acima terem problemas de integração com o ARCore, causando assim um resultado indesejado.

Para essa solução, primeiramente é necessário um projeto Unity, com as configurações corretas dos *plugins* do ARCore e também uma cena que implemente todo o comportamento esperado da funcionalidade. Para facilitar a validação da solução, foi utilizado um projeto de demonstração [2], que já implementa alguns casos de uso de realidade aumentada, inclusive identificação de planos, a princípio horizontais e verticais.

A integração desse projeto Unity com o React Native e com o Flutter se dá através de bibliotecas capazes de exibir esse projeto, uma vez exportado de forma apropriada para Android e IOS, e incluido no código das aplicações móveis como instruem as bibliotecas. Para esse teste, as bibliotecas utilizadas foram a react-native-unity-view [3] e a flutter_unity_widget [4].

Essa abordagem é bem interessante, pois com a Unity ganha-se uma série de interações características de uma *game engine*. No entanto, não é possível a interação dos códigos da aplicação móvel com o projeto Unity, que é pré-compilado. Essa limitação afeta diretamente o dinamismo necessário para a funcionalidade de realidade aumentada do *Classic Blue*. Nesse caso, é fundamental que o quadro exibido seja o escolhido pelo usuário. Por se tratar de uma rede social, novas imagens são recorrentes, dificultando a manutenção do sistema, que toda imagem nova precisaria ser adicionada no projeto Unity, que por sua vez precisaria ser recompilado, necessitando assim de uma atualização no código da aplicação móvel e de um novo lançamento nas lojas de aplicativos.

Além da limitação apresentada acima, com essa solução não foi possível a identificação de planos verticais, assim como na solução anterior. O que comprova a limitação teórica e não um erro de implementação, já que utiliza a própria implementação do Google para Unity.

5.1.3 Reconhecimento de imagem

A última solução implementada e a que funcionou de forma satisfatória foi utilizando a funcionalidade do ARCore de reconhecimento de imagem. Como discutido na seção 4.2.2, existem abordagens eficientes para reconhecimento de imagem, por exemplo, o SIFT, podendo ela ser uma alternativa a identificação de planos verticais.

[1] Link do arcore_flutter_plugin: https://pub.dev/packages/arcore_flutter_plugin

[2] Link do projeto Unity demonstrativo: https://github.com/juicycleff/flutter-unity-arkit-demo

[3] Link para a biblioteca react-native-unity-view: https://github.com/asmadsen/react-native-unity-view

[4] Link para a biblioteca flutter_unity_widget: https://github.com/juicycleff/flutter-unity-view-widget

O primeiro passo dessa implementação foi a criação de uma imagem de referência, que deverá ser identificada na realidade aumentada. Para isso, foi gerado um código QR customizado da *Classic Blue*, como mostra a figura 5.1. Ele foi idealizado para ter duplo propósito, o de identificação na parede, mas, ao mesmo tempo para servir de link para download do aplicativo, caso fosse submetido as lojas de aplicativos. Como a aplicação não foi submetida, o código QR contém os links de download do Bitrise [5] e não das lojas.

Figura 5.1: *Código QR utilizado para o reconhecimento de imagem e também para* download *das aplicações.*

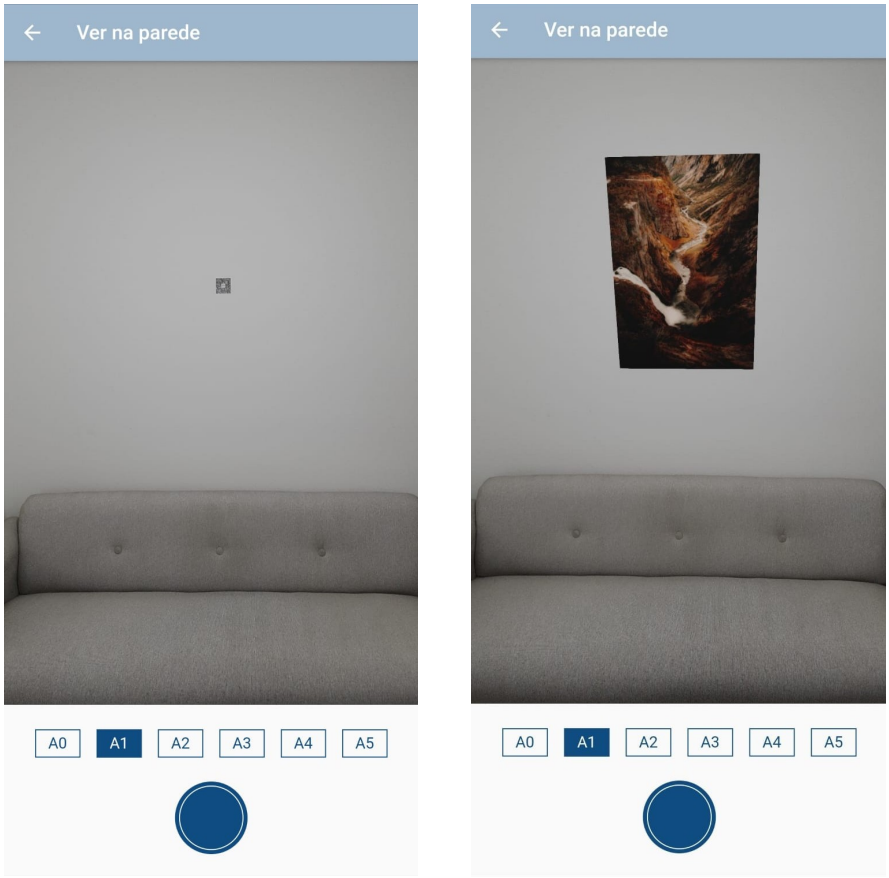
Em seguida, essa imagem foi adicionada em ambos projetos, React Native e Flutter, e a partir de métodos disponibilizados pelas bibliotecas da primeira solução, ela foi incluída no banco de imagens-referência para o reconhecimento de pontos importantes. Feito isso, sempre que a biblioteca identificar o código QR, serão retornadas informações como localização e rotação, que serão utilizadas para a projeção do quadro sobre o código QR de forma realista. A Figura 5.2 mostra um exemplo dessa projeção sendo realizada na prática durante os testes.

Na perspectiva do usuário, a funcionalidade VER NA PAREDE do *Classic Blue* funciona da seguinte forma: primeiramente o usuário entra na aplicação, seleciona uma imagem do FEED; clicando no botão "ver na parede", ele será redirecionado para a próxima tela; nela, o usuário é instruído em como obter o código QR (impressão ou compartilhamento com um segundo dispositivo) e como posicioná-lo na parede; ao finalizar esse tutorial, será redirecionado novamente, mas agora para a tela onde deverá apontar a câmera para código obtido, próxima a ele; após um período de reconhecimento, o quadro será automaticamente projetado na tela, sobre o código QR; e então o usuário conseguirá ver a imagem selecionada como se estivesse na parede.

5.2 Desfecho Sobre as Implementações

Essa experiência não era a imaginada originalmente, porém como vimos na teoria e na prática, o reconhecimento de imagem é a solução mais viável para a projeção de quadros

[5] Link do Bitrise: https://www.bitrise.io/



(a) *Exemplo do código QR na parede.*

(b) *Exemplo do quadro projetado sobre o código QR.*

Figura 5.2: *Exemplo da funcionalidade "ver na parede"na prática.*

numa parede de cor única, considerando a abordagem utilizada pelo ARCore. Dessa forma, são necessárias, além das adaptações técnicas, ajustes nos negócios também, para de forma conjunta facilitar a utilização da funcionalidade, como, por exemplo, enviando para o usuário o código QR já impresso.

Para trabalhos futuros, fica o desafio de buscar por bibliotecas que implementam realidade aumentada utilizando outro método de reconhecimento de imagem, ou de implementar uma biblioteca própria para essa finalidade.

Capítulo 6

Experimento de Usabilidade

Além da análise dos aspectos técnicos realizada no Capítulo 3, neste estudo foi desenvolvido um experimento visando encontrar possíveis diferenças na perspectiva dos usuários. Na Seção 6.1 será apresentada a metodologia utilizada e na Seção 6.2, os resultados obtidos.

6.1 Metodologia

Nesse experimento os participantes foram conduzidos a realizar um conjunto de tarefas em cada uma das aplicações desenvolvidas nesse estudo e foram coletadas informações sobre tempo de conclusão das tarefas, dificuldade de conclusão e também uma avaliação sobre a experiência do usuário.

6.1.1 Descrição do experimento

Inspirado no experimento de Yao *et al.* (2014) e nas instruções para o planejamento de experimentos na área de interação humano-computador do livro escrito por MacKenzie (2013) foi desenvolvido o experimento da seguinte forma. Antes da aplicação do experimento os participantes selecionados foram separados em dois grupos de dez pessoas, A e B, na qual o grupo A testou primeiro a aplicação em Flutter e em seguida a aplicação em React Native, enquanto que o grupo B executou na ordem inversa. Essa separação foi realizada para que o fator aprendizado não influenciasse no resultado, dado que esse é um experimento *within-subjects*, ou seja, o mesmo participante testa cada um dos cenários.

O experimento foi realizado em dois formatos: presencial e *online*. Esses dois formatos representaram, respectivamente, dois cenários de utilização da aplicação, o primeiro em que já é fornecido o código QR (principal cenário idealizado), e outro em que o usuário é instruído na própria aplicação em como obtê-lo. Para o experimento presencial foi utilizado um mesmo dispositivo (Samsung Galaxy Note 10 Lite), já com as aplicações instaladas. No experimento *online*, os dispositivos utilizados (listados a seguir) foram dos próprios participantes, configurados durante o experimento. Esse experimento foi aplicado durante a pandemia de COVID-19 e nem todos os participantes se sentiram confortáveis de realizar

o experimento presencialmente. Considerando o primeiro cenário como o principal e também a preferência de formato de cada participante, foi feita a seguinte divisão: 12 presenciais (6 do grupo A e 6 do grupo B) e 8 *online* (4 do grupo A e 4 do grupo B). Apesar dessa diferença de formato, ela não foi uma variável analisada nesse estudo, que por sua vez focou em comparar a usabilidade da aplicação de forma geral, independente do cenário testado.

Ao iniciar o experimento, foi realizada uma contextualização do estudo em questão, da aplicação Classic Blue e também sobre a realidade aumentada. Depois disso, foi solicitado o preenchimento de um questionário demográfico. Em seguida, foram descritas as quatro tarefas a serem realizadas em cada uma das aplicações, que são:

- **Tarefa 1**: Selecionar imagem;
- **Tarefa 2**: Visualizar imagem na parede;
- **Tarefa 3**: Alterar tamanho da imagem;
- **Tarefa 4**: Salvar na galeria;

Após a descrição das tarefas, foi solicitado que o participante realizasse os seguintes passos:

1. Testar a primeira aplicação, segundo o seu grupo (Primeiro teste);
2. Responder o questionário referente à primeira aplicação;
3. Testar a segunda aplicação, conforme o seu grupo (Segundo teste);
4. Responder o questionário referente à segunda aplicação;

Por fim, foi solicitado o preenchimento de um questionário de considerações finais, na qual o participante pôde dar sua opinião sobre as aplicações testadas, além de um campo para comentários adicionais.

O questionário aplicado aos participantes foi construído utilizando o Google Forms, que pode ser consultado no Apêndice B. A aplicação desse questionário mais a realização dos testes durou em média 27,5 minutos presencialmente e 53,6 minutos *online*. Essa diferença de tempo se deu principalmente pelos passos extras de configuração dos dispositivos utilizados, listados na Tabela 6.1.

Dispositivos	Processador	Cores	RAM (em GB)
Samsung Galaxy Note 10 Lite	Samsung Exynos 9 Octa 9810	8	6
Xiaomi Pocophone F1	Snapdragon 845 Qualcomm SDM845	8	6
Xiaomi Poco X3 NFC	Snapdragon 732G Qualcomm	8	6
Galaxy S10e	Samsung Exynos 9 Octa 9820	8	6
Samsung Galaxy S7 Edge	Samsung Exynos 8 Octa 8890	8	4
Xiaomi Redmi Note 8	Snapdragon 665 Qualcomm SDM665	8	4
Moto G6	Snapdragon 450 Qualcomm SDM450	8	3
Galaxy A10s	Helio P22 MediaTek MT6762	8	2

Tabela 6.1: *Dispositivos utilizados no experimento*

Antes da aplicação desse experimento, foi realizado o teste piloto com um participante, diferente dos utilizados no experimento, com as mesmas características dos demais, também sem conhecimento prévio das aplicações. O objetivo foi garantir que o formato estava adequado e evitar possíveis falhas de aplicação.

6.1.2 Participantes

Os participantes desse experimento foram selecionados a partir da persona definida durante a disciplina de Design de Interfaces, na qual teria as seguintes características:

- jovens que utilizam redes sociais com frequência;
- que tenham interesse em arte e/ou decoração; e
- que não moram com os pais ou responsáveis;

As personas não necessariamente apresentam todas essas características em simultâneo, porém, é desejável que pelo menos duas delas sejam satisfeitas. A partir dessa definição foram selecionadas 20 pessoas que fossem compatíveis com ela.

Na Seção C.1 podem ser consultados os resultados do questionário demográfico aplicado aos participantes.

6.1.3 Métricas

Como citado anteriormente, foram coletadas 3 métricas para comparar as aplicações: tempo de conclusão das tarefas, dificuldade de conclusão e avaliação sobre a experiência do usuário.

A primeira métrica, tempo de conclusão das tarefas, foi calculada indiretamente. Os participantes gravaram a tela do dispositivo durante os testes para ser feita a contabilização dos tempos. As divisões utilizadas foram:

- **Tarefa 1**: Tempo entre o clique em "ENTRAR"e a seleção da primeira imagem;
- **Tarefa 2**: Tempo entre a entrada na tela de detalhe da imagem e a primeira projeção do quadro na parede;
- **Tarefa 3**: Tempo entre a projeção e primeira alteração de tamanho;
- **Tarefa 4**: Tempo entre o clique no botão de fotografar e o salvamento da foto na galeria;

A segunda métrica, dificuldade de conclusão, foi coletada logo após os testes de cada aplicação. Os participantes responderam, para cada tarefa, a dificuldade de conclusão numa escala Likert de 1 a 5, sendo 1 muito fácil e 5 muito difícil.

A última métrica coletada foi a avaliação de experiência do usuário, em que os participantes foram submetidos a um *User Experience Questionnaire* (UEQ) para cada aplicação testada. O UEQ é um questionário proposto por Laugwitz *et al.* (2008), na qual através da avaliação de 26 termos bipolares, é possível quantificar a experiência em seis escalas: atratividade (estética da aplicação), transparência (compreensão e facilidade), eficiência

(velocidade e praticidade), controle (previsibilidade e expectativa), estimulação (interesse e valor), e inovação (criatividade e originalidade).

6.2 Resultados

Considerando as métricas apresentadas acima, a seção de resultados visa apresentar qual das aplicações obteve melhor resultado em cada um delas. Para mais detalhes sobre os resultados consulte o Apêndice C.

6.2.1 Tempo de conclusão das tarefas

Analisando os tempos de conclusão das tarefas coletados durante o experimento obtivemos o gráfico 6.1, que nos mostra que a **Tarefa 2** foi a mais demorada, seguido da **Tarefa 1** que foi cerca de cinco vezes mais rápida. Já a **Tarefa 3** e a **Tarefa 4** foram mais objetivas, com tempo próximo a 8 segundos.

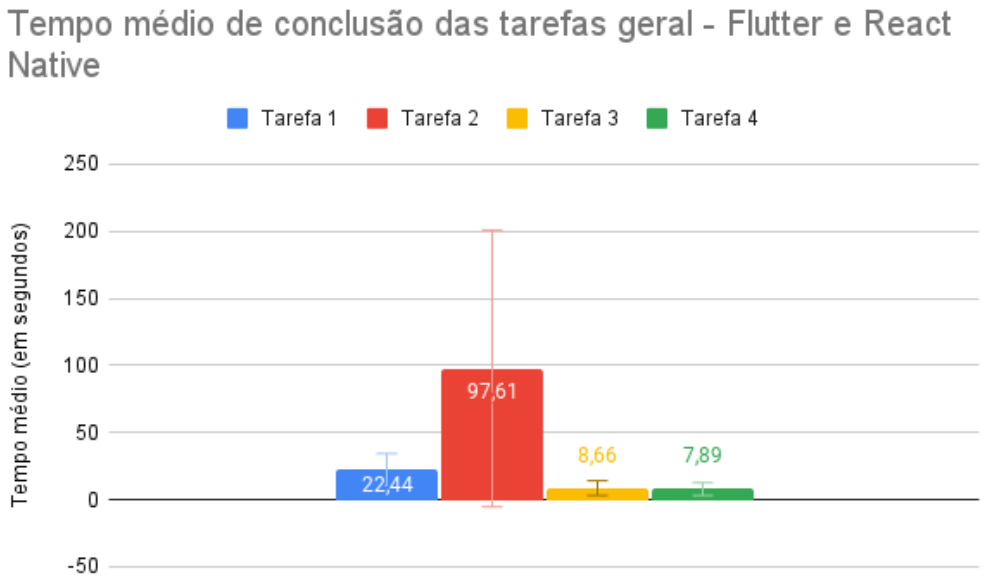


Figura 6.1: *Tempo médio de conclusão das tarefas em geral, considerando os dados dos testes em React Native e em Flutter.*

Os tempos de conclusão no primeiro teste, comparando as duas plataformas, mostraram-se bem próximos na **Tarefa 1** e vantagem para o React Native nas **Tarefa 3** e **Tarefa 4**. A grande diferença foi observada na **Tarefa 2**, na qual os valores divergem em aproximadamente 36 segundos, como mostra o gráfico 6.2. No entanto, essa diferença não representa bem o conjunto de dados observado. De acordo com o método de detecção de *outliers* proposto por Tukey *et al.* (1977), baseado no valor interquartil dos dados, um dos valores (469 segundos) se mostrou acima do limite interno (359 segundos). Removendo esse valor valor que se destacava do conjunto, o novo resultado obtido foi o apresentado no gráfico 6.3, com o Flutter levando uma vantagem de 2 segundos em relação ao React Native.

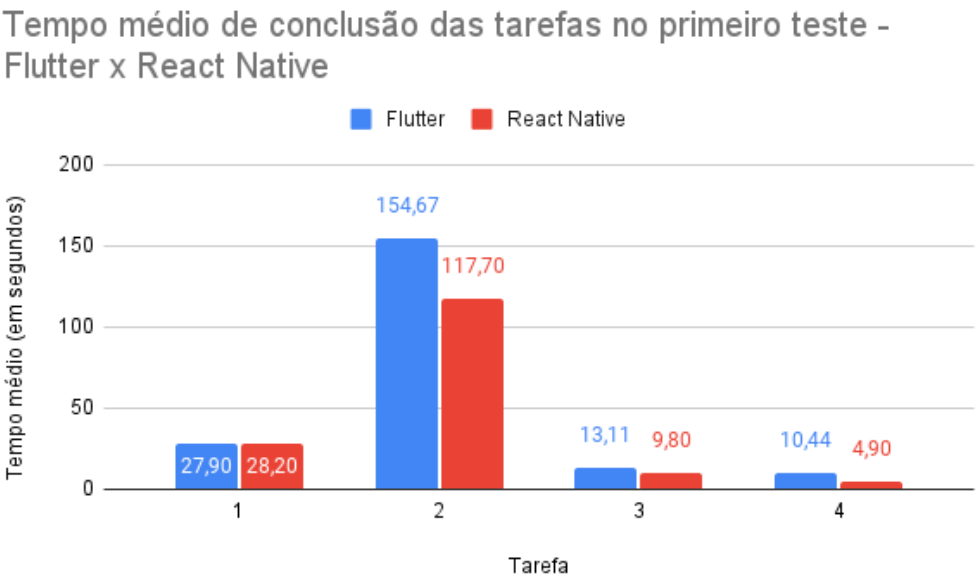


Figura 6.2: *Comparativo do tempo médio de conclusão das tarefas no primeiro teste.*

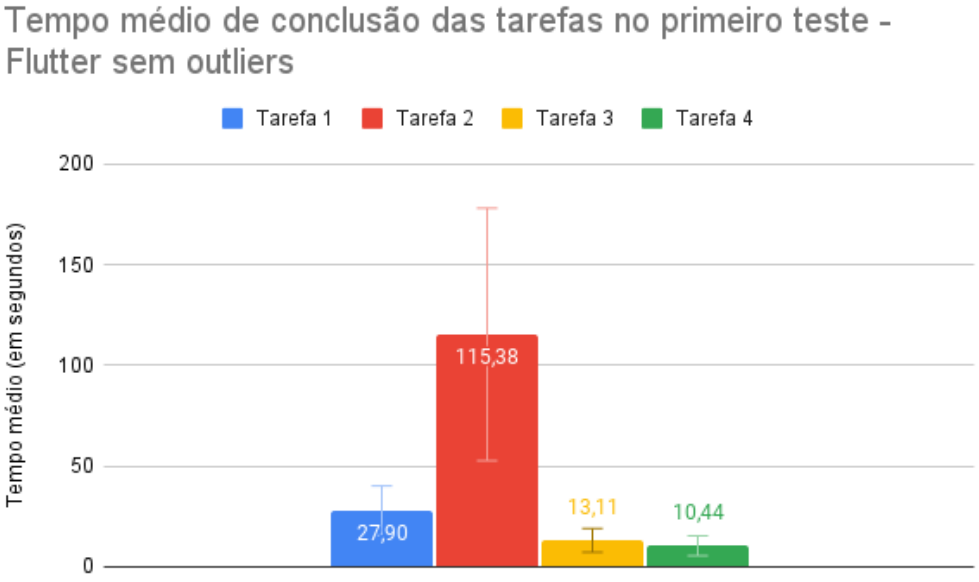


Figura 6.3: *Tempo médio de conclusão das tarefas no primeiro teste realizado em Flutter desconsiderando os* outliers.

No segundo teste, agora desconsiderando o tempo de aprendizagem, observamos que o tempo de conclusão das tarefas, em geral, foi bem menor em relação ao primeiro teste, como esperado. O gráfico 6.4 mostra que o Flutter foi mais rápido na **Tarefa 1**, porém foi mais devagar nas demais, sendo que duas diferenças valem mais atenção. A primeira delas é que o React Native foi em média 70 segundos mais rápido na **Tarefa 2**, porém o conjunto de dados também apresentou *outliers*, utilizando o mesmo método citado anteriormente. Descartando esses valores, obtivemos o gráfico 6.5, que resultaria em uma diferença de apenas 17 segundos, favorável ao React Native. A segunda delas é que na **Tarefa 4** o React Native foi três vezes mais rápido do que o Flutter, dessa vez sem *outliers*, demonstrando uma diferença significativa na perspectiva do usuário.

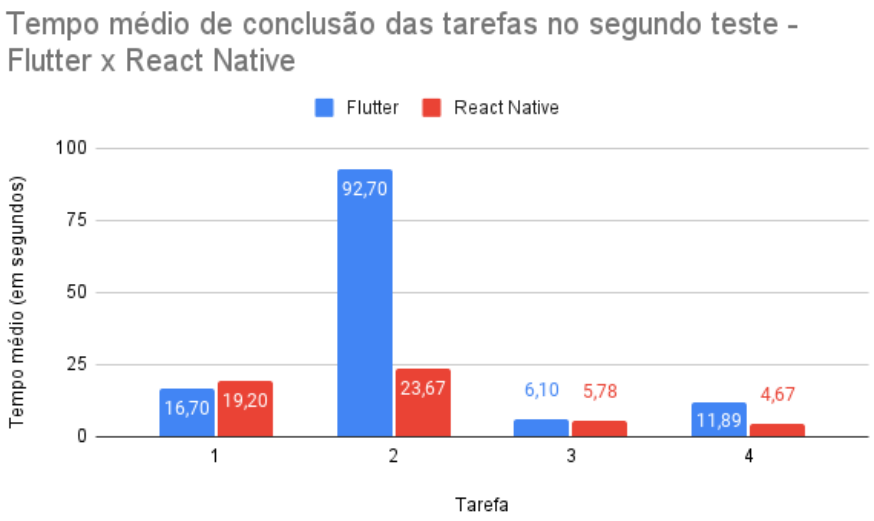


Figura 6.4: *Comparativo do tempo médio de conclusão das tarefas no segundo teste.*

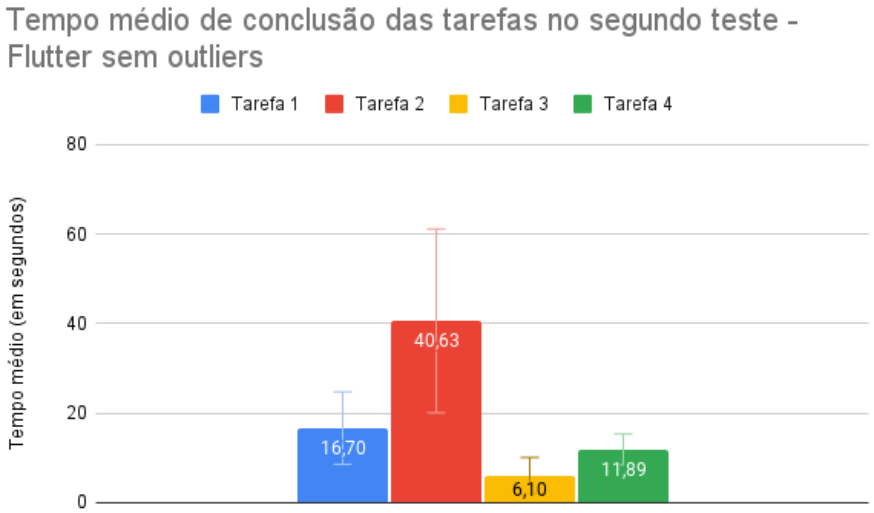


Figura 6.5: *Tempo médio de conclusão das tarefas no segundo teste realizado em Flutter desconsiderando os* outliers.

Analisando todos os dados coletados, as diferenças de tempo entre a **Tarefa 1** e **Tarefa 3** foram pequenas (menos de 2 segundos), quando comparadas as duas plataformas, sendo portanto, uma diferença que não gerou desconforto para os participantes do experimento. Na **Tarefa 2** e **Tarefa 4** o React Native foi 29 segundos e 6 segundos, respectivamente, mais rápido em relação ao Flutter, já desconsiderando os valores *outliers*. Esse valores podem ser consultados no gráfico 6.6.

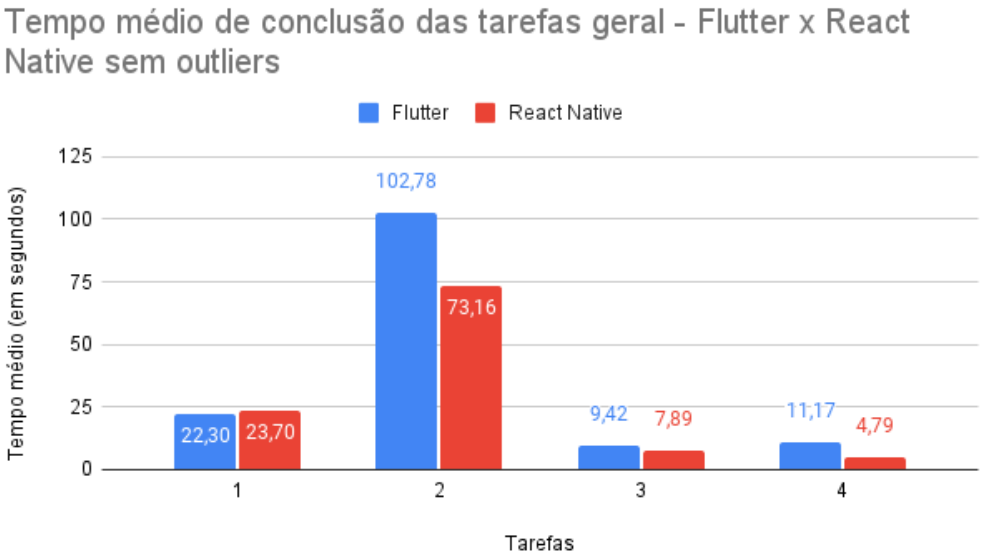


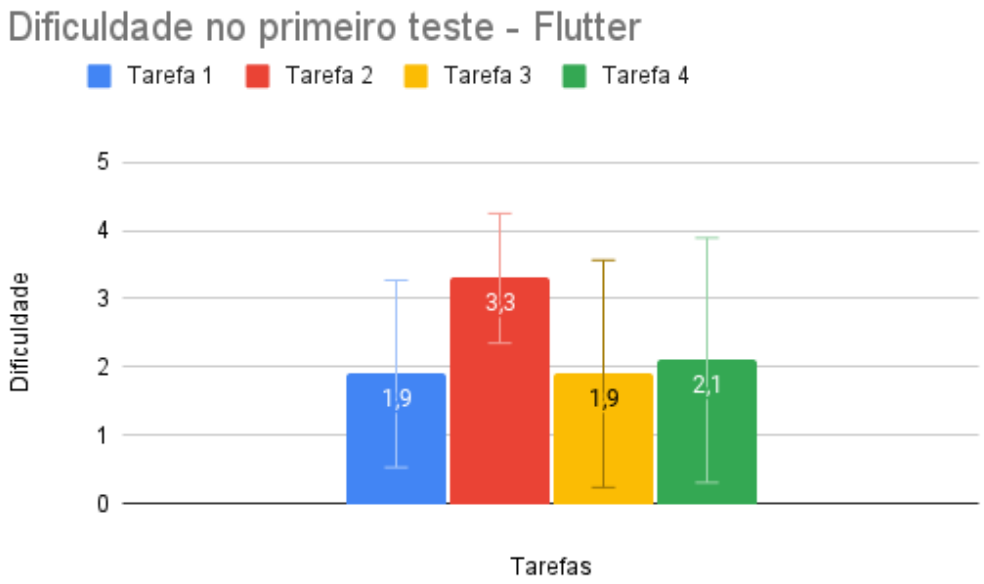
Figura 6.6: *Comparativo do tempo médio de conclusão das tarefas em geral desconsiderando os* outliers.

Durante a análise das gravações foi descoberto que essa diferença aconteceu por três motivos principais: demora maior para detecção do código QR, atraso ao alterar tamanho da imagem, e por uma lentidão ao capturar a imagem projetada. Nos três casos o Flutter foi mais devagar do que o React Native, demonstrando ter algumas limitações inerentes da ferramenta para esses tipos de tarefas.

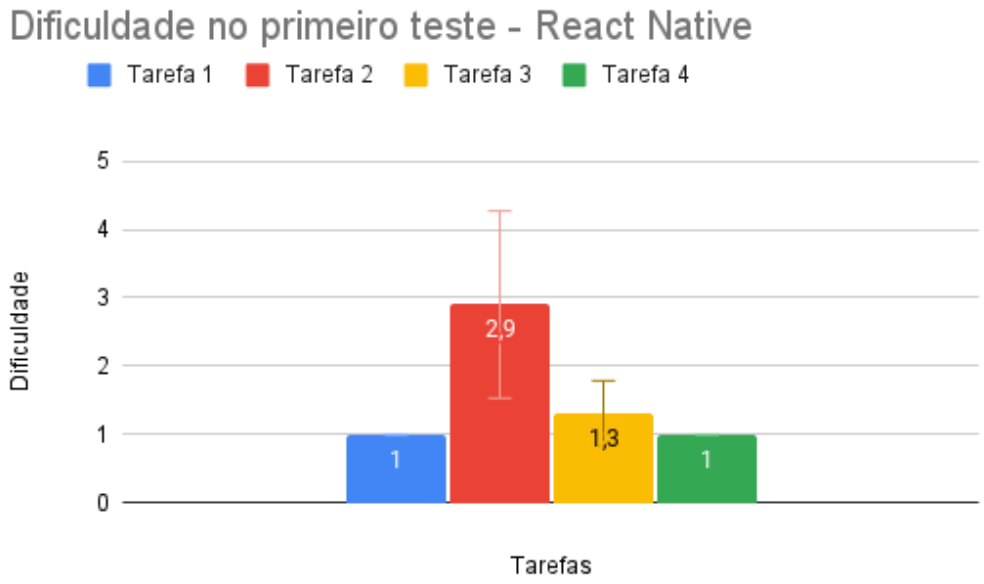
6.2.2 Dificuldade de conclusão das tarefas

A segunda métrica coletada foi em relação à dificuldade percebida pelos participantes em cada um dos testes. Os resultados obtidos foram que a **Tarefa 1**, **Tarefa 3** e **Tarefa 4** foram igualmente fáceis com apenas a **Tarefa 2** apresentando dificuldade maior.

Comparando as aplicações, no primeiro teste o Flutter apresentou na média dificuldade 2 paras a **Tarefa 1**, **Tarefa 3** e **Tarefa 4**, enquanto que a **Tarefa 2** apresentou dificuldade 3. O React Native apresentou para a **Tarefa 1**, **Tarefa 3** e **Tarefa 4**, dificuldade 1 na média e dificuldade 3 para a **Tarefa 2**. Os gráficos 6.7a e 6.7b evidenciam isso.



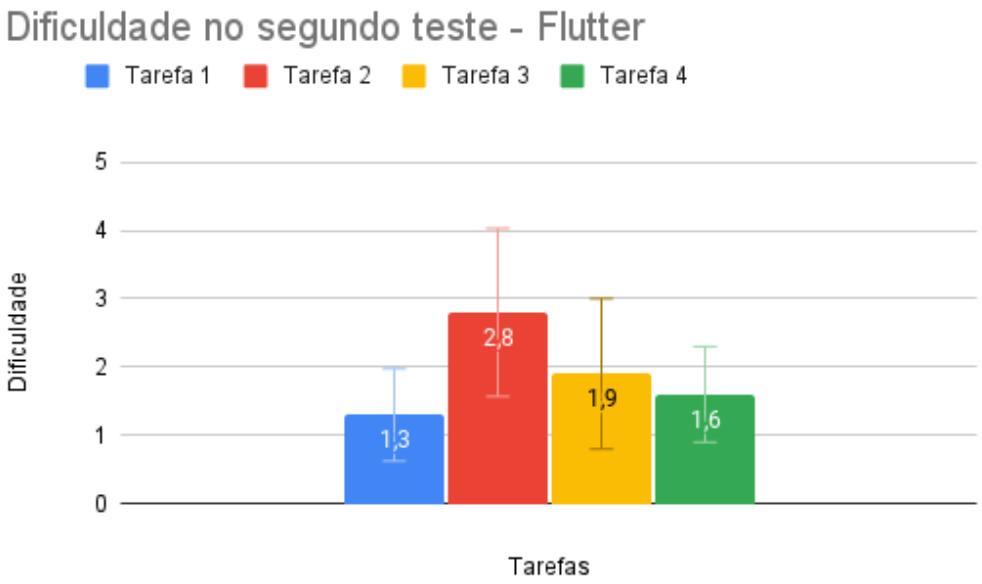
(a) *Dificuldade percebida no primeiro teste em Flutter.*



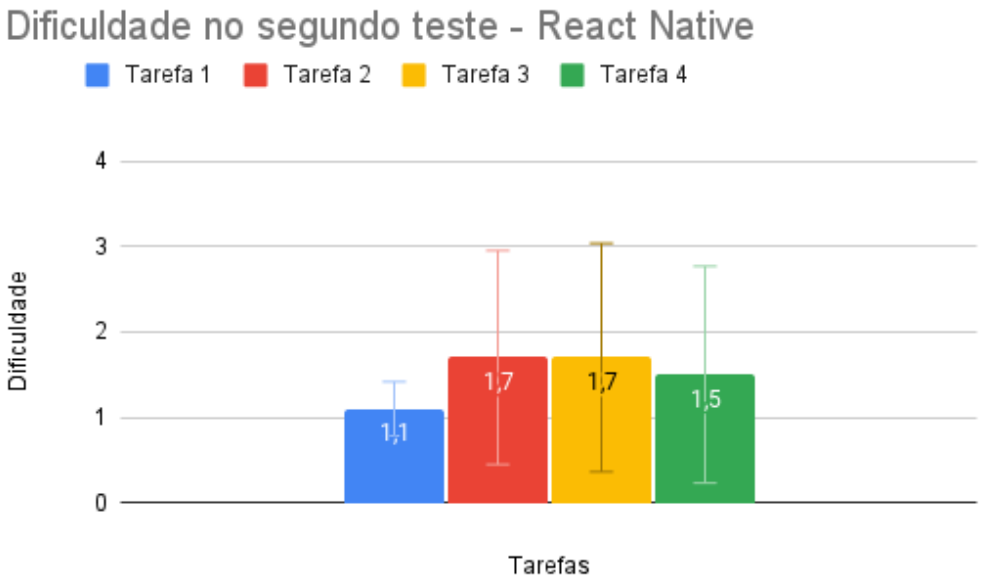
(b) *Dificuldade percebida no primeiro teste em React Native.*

Figura 6.7: *Dificuldade percebida no primeiro teste.*

No segundo teste, apesar dos participantes já conhecerem as tarefas, as dificuldades não apresentaram alterações relevantes. Em Flutter houve uma queda de aproximadamente 0,5 em relação à dificuldade no primeiro teste, com exceção da **Tarefa 3**, que manteve o mesmo valor médio, como mostra o gráfico 6.8a. Em React Native as dificuldades percebidas ficaram em aproximadamente 1,5, menos a **Tarefa 1** que ficou com dificuldade próxima a 1, segundo o gráfico 6.8b.



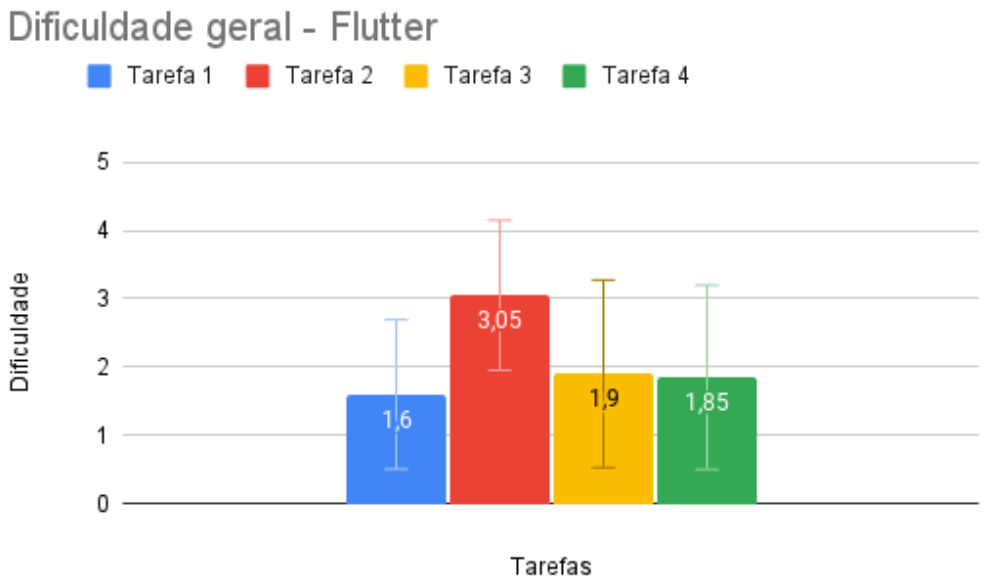
(a) *Dificuldade percebida no segundo teste em Flutter.*



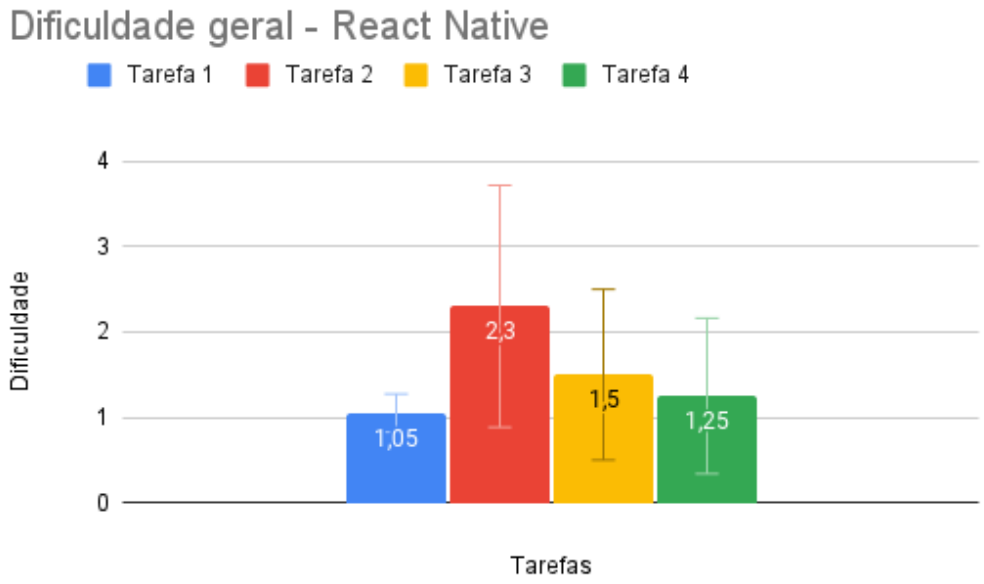
(b) *Dificuldade percebida no segundo teste em React Native.*

Figura 6.8: *Dificuldade percebida no segundo teste.*

Em geral, a dificuldade percebida em React Native foi menor por aproximadamente 0,5 em relação ao Flutter em cada tarefa. Essa diferença pode ser observada nos gráficos 6.9a e 6.9b, que reforçam que os pontos negativos citados na seção 6.2.1 também impactam na dificuldade percebida para completar as tarefas.



(a) *Dificuldade geral percebida em Flutter.*



(b) *Dificuldade geral percebida em React Native.*

Figura 6.9: *Dificuldade percebida no experimento.*

Vale destacar que, durante a aplicação do experimento, um dos participantes não conseguiu completar a **Tarefa 2**, **Tarefa 3** e **Tarefa 4**, nem em React Native e nem em Flutter, pois o dispositivo utilizado (Samsung Galaxy A10s) não era compatível com a versão exigida pelo ARCore. Sendo assim, a funcionalidade de realidade aumentada não funcionou para esse dispositivo, levantando um alerta de que esse tipo de tecnologia pode não ser inclusivo em relação aos dispositivos mais comuns disponíveis no mercado.

6.2.3 Experiência do usuário

A última métrica foi extraída a partir das respostas do UEQ aplicados a cada aplicação testada. Os autores desse questionário, para facilitar sua aplicação e análise disponibilizaram em seu site oficial [1], o questionário em diversos idiomas e também planilhas, nas quais a partir dos dados coletados, já realizam diversas análises implementadas por eles. Tanto o questionário, quanto as planilhas foram utilizados para as análises apresentadas a seguir.

A primeira etapa desse processo de análise foi inserir os dados coletados do questionário em diferentes planilhas, uma considerando apenas dados do primeiro teste, outra apenas do segundo e outra com todos os dados do experimento, separando sempre os dados de cada aplicação. A etapa seguinte utiliza a ferramenta de detecção de dados suspeitos fornecida na planilha para verificar se as respostas de um participante foram aleatórias ou não respondidas de foram coerentes. Essa ferramenta calcula, para cada escala de usabilidade, se houve inconsistências. Caso a ferramenta indique três ou mais escalas inconsistentes, é aconselhado pelos autores a descartar essas respostas. A terceira etapa foi analisar para o tamanho da amostra utilizado a precisão e o erro dos resultados obtidos. Segundo Menditto *et al.* (2007), a precisão pode ser definida como a proximidade entre resultados obtidos sob condição estipulada, nesse caso a proximidade entre a média calculada e o média verdadeira. O erro, como explica Emmert-Streib e Dehmer (2019), pode ser de dois tipos: falso positivo, na qual, em um teste de hipóteses, rejeitamos a hipótese nula e ela é verdadeira, e falso negativo, em que não rejeitamos a hipótese nula dado a hipótese alternativa é verdadeira. Nesse caso estamos considerando o erro como sendo um falso positivo, ou seja, a chance de aceitar que a média verdadeira está fora do intervalo de confiança. Na última etapa foram extraídos os gráficos com os valores de cada escala de usabilidade e também os gráficos que trazem o comparativo com valores de referência. No apêndice C.4 esses gráficos podem ser consultados.

No primeiro teste em Flutter, foram observados respostas inconsistentes de dois participantes, enquanto que em React Native todos os dados foram utilizados. Para o tamanho da amostra utilizado, foi possível atingir uma precisão de 0,5 e erro de 0,1, com exceção de duas escalas em Flutter (transparência e eficiência) e em React Native (transparência e estimulação), que precisariam de uma amostra maior para atingir essa mesma precisão. Comparando as duas plataformas, o gráfico 6.10 mostra que a diferença entre as escalas não foram tão grandes, contudo, fica evidente que o Flutter levou vantagem em quase todos as escalas, com exceção de eficiência e controle, na qual o React Native se saiu melhor. Em relação aos valores de referência, as duas aplicações apresentaram valores em geral entre acima da média e excelente, com apenas algumas escalas estando abaixo da média, que foi o caso da transparência em ambas e o controle em Flutter. Veja os resultados nos gráficos da Figura 6.11.

Observando os dados referentes ao segundo teste, também foram observadas inconsistências nos dados de dois participantes do teste em Flutter, e em React Native, todos os dados foram utilizados novamente. Semelhante ao primeiro teste, a precisão dos dados nesse caso foi de 0,5 com erro de 0,1 e as ressalvas se mantém para o Flutter e React

[1] Link do site oficial do UEQ: https://www.ueq-online.org/

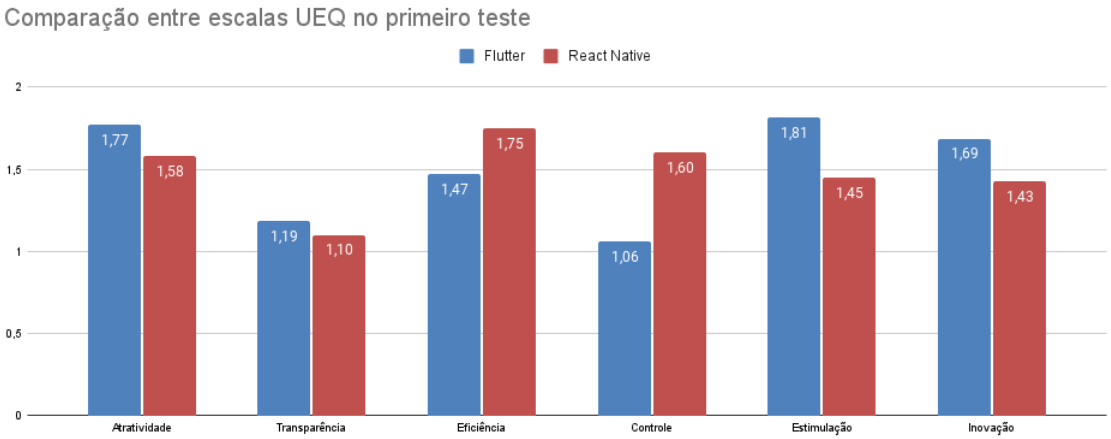
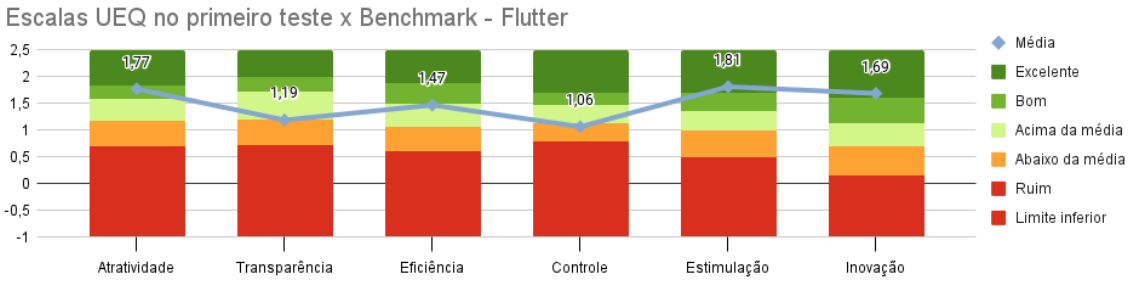
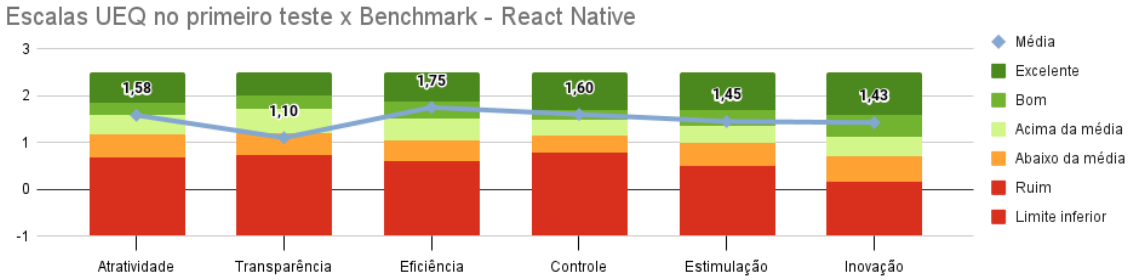


Figura 6.10: *Comparativo entre as escalas UEQ resultantes do primeiro teste.*



(a) *Escalas UEQ resultantes do primeiro teste em Flutter comparadas com valores de referência.*



(b) *Escalas UEQ resultantes do primeiro teste em React Native comparadas com valores de referência.*

Figura 6.11: *Escalas UEQ resultantes do primeiro teste comparadas com valores de referência.*

Native, a menos da escala de transparência que passa a ser contemplada na segunda plataforma. Analisando os resultados apresentados no gráfico 6.12, as diferenças entre as aplicações também não foram tão grandes, porém o React Native apresentou vantagem em praticamente todas as escalas, apenas na escala controle o Flutter teve vantagem. No comparativo com os valores de referência, as duas aplicações apresentaram melhoras, obtendo resultados pelo menos acima da média em cada uma das escalas, como mostra os gráficos da Figura 6.13.

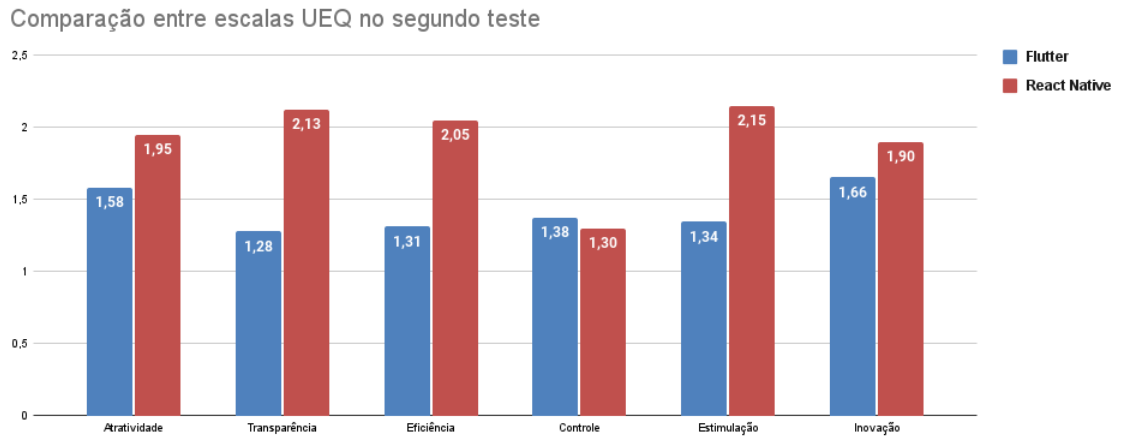
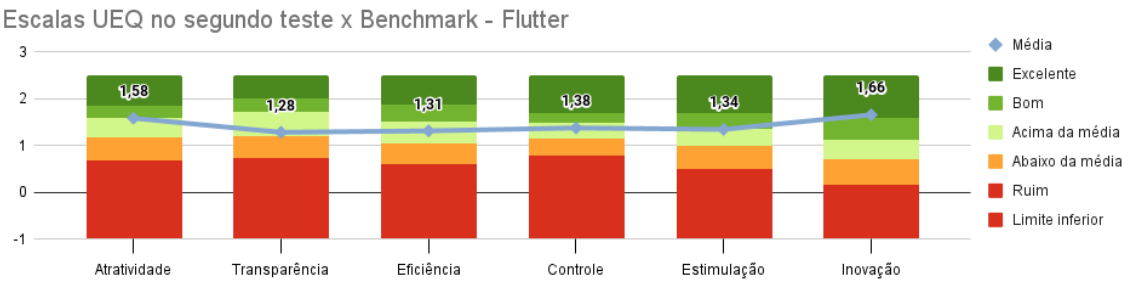
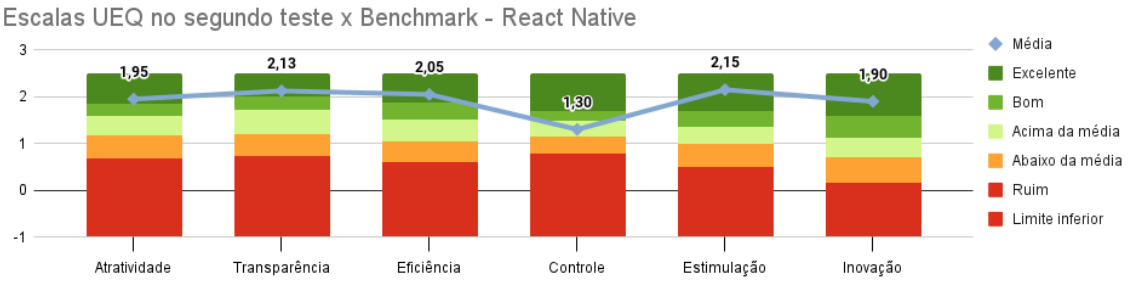


Figura 6.12: *Comparativo entre as escalas UEQ resultantes do segundo teste.*



(a) *Escalas UEQ resultantes do segundo teste em Flutter comparadas com valores de referência.*



(b) *Escalas UEQ resultantes do segundo teste em React Native comparadas com valores de referência.*

Figura 6.13: *Escalas UEQ resultantes do segundo teste comparadas com valores de referência.*

Considerando agora todos os dados consistentes do primeiro e do segundo teste foram realizadas as mesmas análises em cada aplicação. Com uma amostra maior nesse caso,

foi possível obter uma precisão de 0,5 com erro em 0,05 com exceção da transparência em ambas plataformas. Observando os resultados apresentados no gráfico 6.14 fica claro que o React Native apresentou vantagem em todos os aspectos de usabilidade analisados nesse estudo, ficando equiparado apenas em inovação. Por fim, analisando os gráficos da Figura 6.15 vimos que assim como no primeiro e segundo teste, os resultados foram positivos, com a maioria dos aspectos sendo excelente ou pelo menos acima da média.

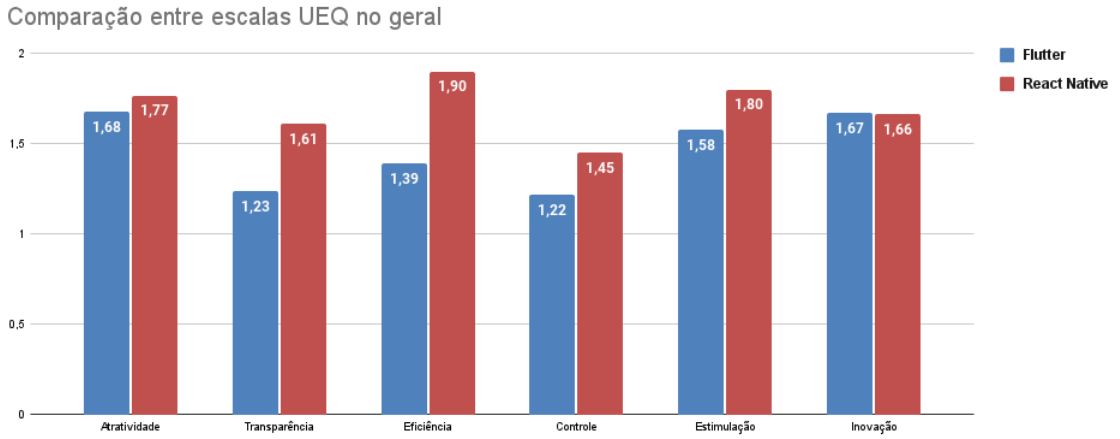
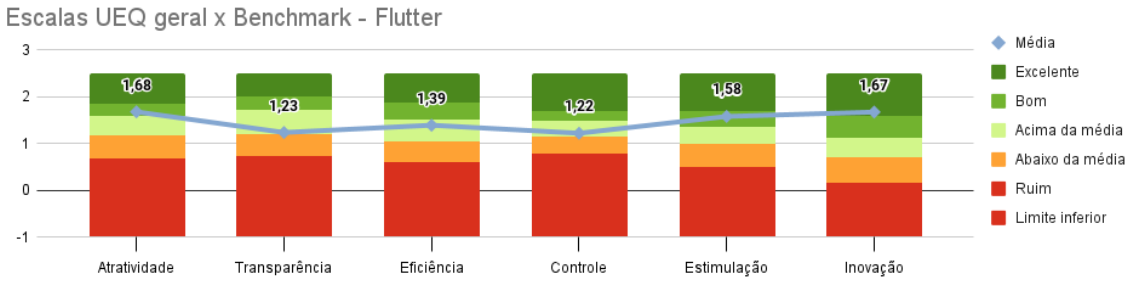
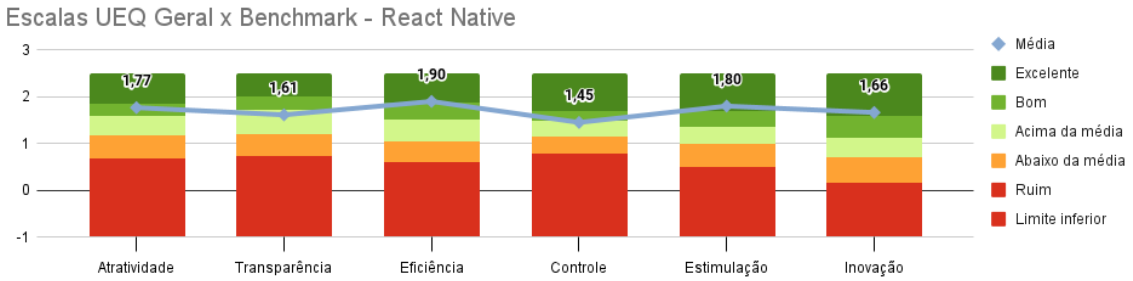


Figura 6.14: *Comparativo entre as escalas UEQ resultantes do experimento.*



(a) *Escalas UEQ resultantes do experimento todo aplicado em Flutter comparadas com valores de referência.*



(b) *Escalas UEQ resultantes do experimento todo aplicado em React Native comparadas com valores de referência.*

Figura 6.15: *Escalas UEQ resultantes do experimento todo comparadas com valores de referência.*

6.3 Considerações dos participantes

Após as análises realizadas, vale apresentar duas considerações sobre experiência relatadas pelos participantes. Em primeiro lugar, ao final do experimento, muitos deles sugeriram uma alternativa para o código QR, por não ser muito prática impressão do código e nem a utilização de um segundo dispositivo. Essa abordagem foi nova para diversos participantes e resultou numa dificuldade de compreensão e consequentemente na grande variação do aspecto transparência, que considera pontos como compreensão, aprendizagem, facilidade e clareza, justificando até mesmo a imprecisão nos resultados obtidos para essa escala. Em segundo lugar, foi perguntado aos participantes, qual das duas aplicações seria provável eles utilizarem e o resultado foi que apenas 3 escolheram o Flutter, enquanto que 17 escolheram o React Native, sendo que as respostas foram às cegas, ou seja, os participantes escolherem sem saber qual foi a tecnologia utilizada em cada teste.

6.4 Conclusão do experimento

Tendo em vista os resultados apresentados na Seção 6.2 e as considerações da Seção 6.3 podemos concluir que o React Native levou vantagem em relação ao Flutter no aspecto de usabilidade. O React Native se mostrou mais rápido, mais fácil para resolver as tarefas e também obteve melhores notas em praticamente todas as escalas de usabilidade analisadas. Portanto, essa ferramenta seria a mais indicada pensando em experiência de usuário para aplicações com realidade aumentada.

Capítulo 7

Conclusão

Durante esta pesquisa foram discutidos diversos aspectos referentes às duas ferramentas de desenvolvimento híbrido móvel (React Native e Flutter) para auxiliar outros desenvolvedores a escolherem uma delas para o desenvolvimento de aplicações com realidade aumentada. A seguir discutiremos sobre as descobertas feitas durante o desenvolvimento desse trabalho.

Em primeiro lugar, vimos que, de forma não restrita as ferramentas de desenvolvimento utilizadas, a utilização do ARCore para funcionalidades que necessitam do reconhecimento de planos verticais pode ser não ser muito indicada, pois ela possui uma limitação no reconhecimento de imagens sem contraste, como é o caso de paredes sem textura e de cor única. Essa limitação leva os desenvolvedores e *stakeholders* a buscarem alternativas como a apresentada nesse estudo, que envolve tanto tecnologia, como negócios.

Em segundo lugar, comparando as linguagens nos aspectos técnicos, elas apresentam diversas semelhanças em relação a conceitos como reatividade e composição e diferem em outros como paradigma principal de desenvolvimento e estrutura de código. A lógica de desenvolvimento é bem semelhante nos dois casos, porém a identificação do React Native com outras ferramentas de desenvolvimento web facilitam a adaptação inicial de quem tem essa perícia, enquanto que o Flutter se aproxima das ferramentas nativas, facilitando a adaptação nesse caso. Por fim, outro ponto destacado nesse estudo foi em relação à maturidade da comunidade e o quanto isso pode impactar na decisão das ferramentas, na qual a comunidade React Native demonstrou estar mais madura em relação a comunidade Flutter.

Em terceiro lugar, os resultados do experimento aplicado nesse estudo mostram que o React Native foi mais rápido, demostrou ser mais fácil para a resolução das tarefas propostas e também apresentou melhores valores em cinco dos seis aspectos de usabilidade analisados (atratividade, transparência, eficiência, controle, estimulação e inovação). Então pensando em experiência de usuário para aplicações que contém realidade aumentada, o React Native pode ser o mais indicado.

Considerando os pontos citados acima, podemos concluir que o React Native levou vantagem tanto nos aspectos técnicos, quanto nos aspectos de usabilidade analisados nesse estudo, tornando-o a ferramenta de desenvolvimento híbrido móvel mais indicada para o

desenvolvimento de aplicações com funcionalidades de realidade aumentada. O Flutter, apesar de suas qualidades, não seria a ferramenta mais indicada para esse tipo de aplicação, pois apresentou limitações técnicas consequentes de uma comunidade pouco madura, que acabam sendo refletidas na experiência do usuário.

Os resultados desse estudo podem sofrer alterações nos próximos anos com o amadurecimento da comunidade Flutter e com uma difusão maior da ferramenta no ambiente acadêmico e também corporativo. Então para trabalhos futuros, algumas sugestões são: validação do estudo também em IOS, que não foi contemplado nesse estudo; comparação de outros recursos das ferramentas, como, por exemplo, persistência de dados; e também a revalidação desse comparativo daqui a alguns anos, para verificar se houve alguma mudança em relação aos resultados apresentados nesse trabalho.

Apêndice A

Ferramentas de Desenvolvimento Auxiliares

O processo de desenvolvimento de código envolve uma série de etapas. Vithani e Kumar, 2014 propuseram um ciclo de desenvolvimento para aplicações móveis, o MADLC (*Mobile Application Development Lifecycle*), que possui as seguintes etapas: Identificação, *Design*, Desenvolvimento, Prototipação, Teste, Entrega e Manutenção. Para isso é inevitável o uso de ferramentas auxiliares. Por exemplo, durante esse estudo foram utilizadas ferramentas de design e prototipação, de versionamento, de edição de código, de testes e de integração e entrega continuas.

A seguir serão apresentadas as ferramentas utilizadas, que foram:

A.1 Marvel

Como citado no capítulo 2, a ferramenta de design utilizada foi o Marvel[1]. Ele é uma ferramenta que permite um rápido processo de prototipação, teste e entrega do resultado final para o time. Ele foi escolhido principalmente por sua funcionalidade de construir um protótipo navegável a partir de desenhos no papel. Isso foi importante principalmente nos estágios iniciais da idealização do aplicativo. Mas também mostrou-se uma ferramenta intuitiva e poderosa nas etapas seguintes.

A.2 Gitlab

O Gitlab[2] é a ferramenta de DevOps mais utilizada do mercado, com uma estimativa de mais de 30 milhões de usuários registrados (Gitlab, 2021). DevOps, segundo Jabbari *et al.*, 2016, é uma metodologia que faz a ponte entre desenvolvimento e operações, com ênfase na comunicação e colaboração, integração continua, garantia de qualidade, entrega continua com implantação automática utilizando uma série de práticas de desenvolvimento.

[1] Link do Marvel: https://marvelapp.com/. Acesso em 15 Ago 2021

[2] Link do Gitlab: https://about.gitlab.com. Acesso em 15 Ago 2021

O grande motivo dessa ferramenta ser a mais utilizada deve-se ao fato dela fazer essa ponte de forma completa, através do versionador de código integrado ao serviço de integração e entrega contínua e da quantidade de possíveis integrações com plataformas externas.

Para esse projeto, o Gitlab foi útil principalmente pela fácil integração com a plataforma Bitrise, que será apresentada na subseção A.5. Essa integração foi feita utilizando uma estratégia de acionamento de gatilhos do Bitrise com base em atualizações das duas principais ramificações de código utilizadas, *master* e *develop*. O desenvolvimento foi feito sempre com base no código da ramificação *develop*, que acionava um gatilho específico de desenvolvimento. A ramificação *master*, por sua vez, era atualizada sempre que houvesse uma versão estável da aplicação e gerasse valor para o usuário, acionando consequentemente outro gatilho configurado especificamente para produção.

A.3 Visual Studio Code

A ferramenta utilizada para edição de código foi o Visual Studio Code (VS Code)[3], que é um leve, mas poderoso editor de código-fonte desenvolvido pela Microsoft. Uma de suas principais vantagens, além de ser leve, como citado anteriormente, é que ele possui diversas integrações poderosas (chamadas extensões). Essas extensões facilitam o desenvolvimento, pois agregam diversas funcionalidades durante a escrita do código e aumentam a produtividade. Por exemplo, foram utilizadas extensões de formatação de código, de integrações com ferramentas de teste, de integração com versionadores de código, de IntelliSense (conhecido como assistente de código).

Vale a pena comentar que foi analisada a possibilidade de utilizar o Android Studio[4], que é a ferramenta de edição de código oficial para desenvolvimento Android (Google, 2021a). No entanto, ela exige bastante dos recursos do computador para apresentar um bom desempenho e apesar de também possuir algumas integrações como a do versionador e também assistente de código, não são bem construídas como no VS Code. A grande vantagem do Android Studio é a integração com o emulador Android, que permite testar o aplicativo diretamente do computador sem precisar de um dispositivo. Contudo, a funcionalidade de realidade aumentada depende de câmera, por isso não faz sentido usar um emulador e sim o próprio dispositivo. Sendo assim, a utilização dessa ferramenta se tornou desnecessária.

A.4 Ferramentas de teste

Para os testes foram utilizadas duas ferramentas em cada linguagem, elas são diferentes, porém, com propósitos parecidos. Para Flutter foi utilizado o conjunto flutter_test[5] e o

[3] Link do Visual Studio Code: https://code.visualstudio.com/docs. Acesso em 15 Ago 2021

[4] Link do Android Studio: https://developer.android.com/studio. Acesso em 15 Ago 2021

[5] Link do flutter_test: https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html. Acesso em 15 Ago 2021

mockito[6], enquanto que para React Native foi utilizado o Jest[7] e a Testing Library[8]. Tanto o flutter_test quanto a Testing Library foram utilizados para emular componentes visuais e testar as possíveis interações. Já o mockito e o Jest foram utilizados para simular requisições e bibliotecas no ambiente de teste.

Essas ferramentas foram utilizadas para dois tipos de testes, testes unitários e de integração. Os testes unitários foram implementados para validar principalmente as regras de negócio da aplicação e os testes de integração foram utilizados para validar de forma automatizada as funcionalidades existentes.

A.5 Bitrise

O Bitrise[9] é uma ferramenta de integração e entrega continua para aplicações móveis. Ela oferece mais de 300 passos e integrações para conectar processos e serviços que geralmente são necessários durante o fluxo de desenvolvimento móvel. O ponto forte dessa plataforma é a fácil customização desses passos e integrações para a criação de fluxos específicos para cada necessidade.

Nesse caso, foram criados dois fluxos principais, um para teste e outro para implantação. O fluxo de teste foi configurado para ser acionado a partir de uma atualização da ramificação *develop* e executar a rotina de testes implementadas. Já o fluxo de implantação foi customizado para ser acionado apenas com a atualização da *master*. Esse fluxo além de executar os testes, também executa os passos necessários para a geração do *Android Application Package* (APK), arquivo que depois pode ser utilizado para disponibilização na loja de aplicativos, Google Play. Como a geração da APK é último passo em que as duas linguagens se diferem, o fluxo de implantação foi configurado até esse ponto apenas.

[6] Link do mockito: https://pub.dev/packages/mockito. Acesso em 15 Ago 2021

[7] Link do Jest: https://jestjs.io/docs/tutorial-react-native. Acesso em 15 Ago 2021

[8] Link do Testing Library: https://testing-library.com/docs/react-native-testing-library/intro/. Acesso em 15 Ago 2021

[9] Link do Bitrise: https://www.bitrise.io/. Acesso em 15 Ago 2021

Apêndice B

Questionário Utilizado no Experimento de Usabilidade

Pesquisa de Usabilidade da Classic Blue

Essa pesquisa tem como objetivo analisar a usabilidade dos aplicativos construídos durante o desenvolvimento do TCC do aluno Victor Martins João, estudante de Ciência da Computação (IME-USP). Essa pesquisa também fará parte do estudo.

A pesquisa é dividia em quatro seções:
1 - Dados do participante: Informações sobre o entrevistado relevantes para a pesquisa;
2 - Usabilidade do primeiro aplicativo: Respostas sobre a usabilidade do primeiro aplicativo testado pelo participante;
3 - Usabilidade do segundo aplicativo: Respostas sobre a usabilidade do segundo aplicativo testado pelo participante;
4 - Comentários finais: Respostas finais da pesquisa e espaço livre para o participante dar sua opinião sobre a pesquisa;

Muito obrigado por participar :)

*Obrigatório

Dados do participante

Nessa seção você deve preencher as seguintes informações sobre você, relevantes para a pesquisa.

1. Número do participante *

2. Idade *

3. Gênero *

Marcar apenas uma oval.

- ◯ Masculino
- ◯ Feminino
- ◯ Prefiro não dizer
- ◯ Outro: _____

4. Mora com seus pais ou responsáveis? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Outro: _____

5. Como você classificaria seu interesse em arte e decoração? *

Marcar apenas uma oval.

	1	2	3	4	5	
Não tenho interesse	◯	◯	◯	◯	◯	Tenho muito interesse

6. Quantas horas você utiliza redes sociais por semana? *

Marcar apenas uma oval.

- ◯ Até 1 hora
- ◯ De 1 a 3 horas
- ◯ De 3 a 7 horas
- ◯ Mais de 7 horas

7. Quais redes sociais você mais utiliza? *

Marque todas que se aplicam.

- ☐ Instagram
- ☐ Facebook
- ☐ Twitter
- ☐ Pinterest

Outro: ☐ _____

8. Você já teve alguma experiência com Realidade Aumentada? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Não sei

Usabilidade do primeiro aplicativo

Nesta seção você deverá responder as perguntas a seguir referentes ao primeiro aplicativo testado.

9. Como você classificaria a dificuldade para completar a tarefa 1 no primeiro aplicativo (Selecionar imagem)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

10. Como você classificaria a dificuldade para completar a tarefa 2 no primeiro aplicativo (Visualizar imagem na parede)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

11. Como você classificaria a dificuldade para completar a tarefa 3 no primeiro aplicativo (Alterar tamanho da imagem)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

12. Como você classificaria a dificuldade para completar a tarefa 4 no primeiro aplicativo (Salvar na galeria)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

13. Como você classificaria o primeiro aplicativo em relação aos termos Desagradável/Agradável? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Desagradável	◯	◯	◯	◯	◯	◯	◯	Agradável

14. Como você classificaria o primeiro aplicativo em relação aos termos Incompreensível/Compreensível? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Incompreensível	◯	◯	◯	◯	◯	◯	◯	Compreensível

15. Como você classificaria o primeiro aplicativo em relação aos termos Criativo/Sem criatividade? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Criativo	◯	◯	◯	◯	◯	◯	◯	Sem criatividade

16. Como você classificaria o primeiro aplicativo em relação aos termos De Fácil aprendizagem/De difícil aprendizagem? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
De Fácil aprendizagem	◯	◯	◯	◯	◯	◯	◯	De difícil aprendizagem

17. Como você classificaria o primeiro aplicativo em relação aos termos Valioso/Sem valor? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Valioso	◯	◯	◯	◯	◯	◯	◯	Sem valor

18. Como você classificaria o primeiro aplicativo em relação aos termos Aborrecido/Excitante? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Aborrecido	◯	◯	◯	◯	◯	◯	◯	Excitante

19. Como você classificaria o primeiro aplicativo em relação aos termos Desinteressante/Interessante? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Desinteressante	◯	◯	◯	◯	◯	◯	◯	Interessante

20. Como você classificaria o primeiro aplicativo em relação aos termos Imprevisível/Previsível? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Imprevisível	◯	◯	◯	◯	◯	◯	◯	Previsível

21. Como você classificaria o primeiro aplicativo em relação aos termos Rápido/Lento? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Rápido	◯	◯	◯	◯	◯	◯	◯	Lento

22. Como você classificaria o primeiro aplicativo em relação aos termos Original/Convencional? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Original	◯	◯	◯	◯	◯	◯	◯	Convencional

23. Como você classificaria o primeiro aplicativo em relação aos termos Obstrutivo/Condutor? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Obstrutivo	◯	◯	◯	◯	◯	◯	◯	Condutor

24. Como você classificaria o primeiro aplicativo em relação aos termos Bom/Mau? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Bom	◯	◯	◯	◯	◯	◯	◯	Mau

25. Como você classificaria o primeiro aplicativo em relação aos termos Complicado/Fácil? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Complicado	◯	◯	◯	◯	◯	◯	◯	Fácil

26. Como você classificaria o primeiro aplicativo em relação aos termos Desinteressante/Atrativo? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Desinteressante	◯	◯	◯	◯	◯	◯	◯	Atrativo

27. Como você classificaria o primeiro aplicativo em relação aos termos Comum/Vanguardista? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Comum	◯	◯	◯	◯	◯	◯	◯	Vanguardista

28. Como você classificaria o primeiro aplicativo em relação aos termos Incômodo/Cômodo? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Incômodo	◯	◯	◯	◯	◯	◯	◯	Cômodo

29. Como você classificaria o primeiro aplicativo em relação aos termos Seguro/Inseguro? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Seguro	◯	◯	◯	◯	◯	◯	◯	Inseguro

30. Como você classificaria o primeiro aplicativo em relação aos termos Motivante/Desmotivante? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Motivante	◯	◯	◯	◯	◯	◯	◯	Desmotivante

31. Como você classificaria o primeiro aplicativo em relação aos termos Atende as expectativas/Não atende as expectativas? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Atende as expectativas	◯	◯	◯	◯	◯	◯	◯	Não atende as expectativas

32. Como você classificaria o primeiro aplicativo em relação aos termos Ineficiente/Eficiente? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Ineficiente	◯	◯	◯	◯	◯	◯	◯	Eficiente

33. Como você classificaria o primeiro aplicativo em relação aos termos Evidente/Confuso? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Evidente	◯	◯	◯	◯	◯	◯	◯	Confuso

34. Como você classificaria o primeiro aplicativo em relação aos termos Impraticável/Prático? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Impraticável	◯	◯	◯	◯	◯	◯	◯	Prático

35. Como você classificaria o primeiro aplicativo em relação aos termos Organizado/Desorganizado? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Organizado	◯	◯	◯	◯	◯	◯	◯	Desorganizado

36. Como você classificaria o primeiro aplicativo em relação aos termos Atraente/Feio? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Atraente	◯	◯	◯	◯	◯	◯	◯	Feio

37. Como você classificaria o primeiro aplicativo em relação aos termos Simpático/Antipático? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Simpático	◯	◯	◯	◯	◯	◯	◯	Antipático

38. Como você classificaria o primeiro aplicativo em relação aos termos Conservador/Inovador? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Conservador	◯	◯	◯	◯	◯	◯	◯	Inovador

Usabilidade do segundo aplicativo

Nesta seção você deverá responder as perguntas a seguir referentes ao segundo aplicativo testado.

39. Como você classificaria a dificuldade para completar a tarefa 1 no segundo aplicativo (Selecionar imagem)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

40. Como você classificaria a dificuldade para completar a tarefa 2 no segundo aplicativo (Visualizar imagem na parede)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

41. Como você classificaria a dificuldade para completar a tarefa 3 no segundo aplicativo (Alterar tamanho da imagem)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

42. Como você classificaria a dificuldade para completar a tarefa 4 no segundo aplicativo (Salvar na galeria)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Muito fácil	◯	◯	◯	◯	◯	Muito difícil

43. Como você classificaria o segundo aplicativo em relação aos termos Desagradável/Agradável? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Desagradável	◯	◯	◯	◯	◯	◯	◯	Agradável

44. Como você classificaria o segundo aplicativo em relação aos termos Incompreensível/Compreensível? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Incompreensível	◯	◯	◯	◯	◯	◯	◯	Compreensível

45. Como você classificaria o segundo aplicativo em relação aos termos Criativo/Sem criatividade? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Criativo	◯	◯	◯	◯	◯	◯	◯	Sem criatividade

46. Como você classificaria o segundo aplicativo em relação aos termos De Fácil aprendizagem/De difícil aprendizagem? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
De Fácil aprendizagem	◯	◯	◯	◯	◯	◯	◯	De difícil aprendizagem

47. Como você classificaria o segundo aplicativo em relação aos termos Valioso/Sem valor? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Valioso	◯	◯	◯	◯	◯	◯	◯	Sem valor

48. Como você classificaria o segundo aplicativo em relação aos termos Aborrecido/Excitante? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Aborrecido	◯	◯	◯	◯	◯	◯	◯	Excitante

49. Como você classificaria o segundo aplicativo em relação aos termos Desinteressante/Interessante? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Desinteressante	◯	◯	◯	◯	◯	◯	◯	Interessante

50. Como você classificaria o segundo aplicativo em relação aos termos Imprevisível/Previsível? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Imprevisível	◯	◯	◯	◯	◯	◯	◯	Previsível

51. Como você classificaria o segundo aplicativo em relação aos termos Rápido/Lento? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Rápido	◯	◯	◯	◯	◯	◯	◯	Lento

52. Como você classificaria o segundo aplicativo em relação aos termos Original/Convencional? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Original	◯	◯	◯	◯	◯	◯	◯	Convencional

53. Como você classificaria o segundo aplicativo em relação aos termos Obstrutivo/Condutor? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Obstrutivo	◯	◯	◯	◯	◯	◯	◯	Condutor

54. Como você classificaria o segundo aplicativo em relação aos termos Bom/Mau? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Bom	◯	◯	◯	◯	◯	◯	◯	Mau

55. Como você classificaria o segundo aplicativo em relação aos termos Complicado/Fácil? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Complicado	◯	◯	◯	◯	◯	◯	◯	Fácil

56. Como você classificaria o segundo aplicativo em relação aos termos Desinteressante/Atrativo? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Desinteressante	◯	◯	◯	◯	◯	◯	◯	Atrativo

57. Como você classificaria o segundo aplicativo em relação aos termos Comum/Vanguardista? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Comum	◯	◯	◯	◯	◯	◯	◯	Vanguardista

58. Como você classificaria o segundo aplicativo em relação aos termos Incômodo/Cômodo? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Incômodo	◯	◯	◯	◯	◯	◯	◯	Cômodo

59. Como você classificaria o segundo aplicativo em relação aos termos Seguro/Inseguro? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Seguro	◯	◯	◯	◯	◯	◯	◯	Inseguro

60. Como você classificaria o segundo aplicativo em relação aos termos Motivante/Desmotivante? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Motivante	◯	◯	◯	◯	◯	◯	◯	Desmotivante

61. Como você classificaria o segundo aplicativo em relação aos termos Atende as expectativas/Não atende as expectativas? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Atende as expectativas	◯	◯	◯	◯	◯	◯	◯	Não atende as expectativas

62. Como você classificaria o segundo aplicativo em relação aos termos Ineficiente/Eficiente? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Ineficiente	◯	◯	◯	◯	◯	◯	◯	Eficiente

63. Como você classificaria o segundo aplicativo em relação aos termos Evidente/Confuso? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Evidente	◯	◯	◯	◯	◯	◯	◯	Confuso

64. Como você classificaria o segundo aplicativo em relação aos termos Impraticável/Prático? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Impraticável	◯	◯	◯	◯	◯	◯	◯	Prático

65. Como você classificaria o segundo aplicativo em relação aos termos Organizado/Desorganizado? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Organizado	◯	◯	◯	◯	◯	◯	◯	Desorganizado

66. Como você classificaria o segundo aplicativo em relação aos termos Atraente/Feio? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Atraente	◯	◯	◯	◯	◯	◯	◯	Feio

67. Como você classificaria o segundo aplicativo em relação aos termos Simpático/Antipático? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Simpático	◯	◯	◯	◯	◯	◯	◯	Antipático

68. Como você classificaria o segundo aplicativo em relação aos termos Conservador/Inovador? *

Marcar apenas uma oval.

	1	2	3	4	5	6	7	
Conservador	◯	◯	◯	◯	◯	◯	◯	Inovador

Comentários finais

Após testar os dois aplicativos, responda da melhor forma as perguntas abaixo. Por fim, fique livre para adicionar comentários sobre a entrevista, sobre o Classic Blue e sobre os aplicativos testados :)

69. Você utilizaria essa aplicação no seu dia a dia? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Talvez

70. Qual dos dois métodos de posicionamento do código QR você utilizou? *

Marcar apenas uma oval.

- ◯ Código QR impresso
- ◯ Código QR em um segundo dispositivo

71. Qual dos dois aplicativos testados seria mais provável você utilizar? *

Marcar apenas uma oval.

- ◯ Primeiro aplicativo
- ◯ Segundo aplicativo

72. Na sua opinião, teria algo a ser melhorado? Se sim, o quê? *

73. Adicione aqui seus comentários adicionais

Apêndice C

Resultados do experimento

Esse apêndice é destinado aos resultados do experimento aplicado durante esse estudo. Nas seções a seguir serão apresentados os resultados do questionário demográfico, do tempo de conclusão das tarefas, da dificuldade percebida pelos participantes e do relato da experiência deles.

C.1 Questionário demográfico

A seguir são apresentados os gráficos referentes a cada uma das perguntas demográficas aplicadas aos participantes durante o experimento.

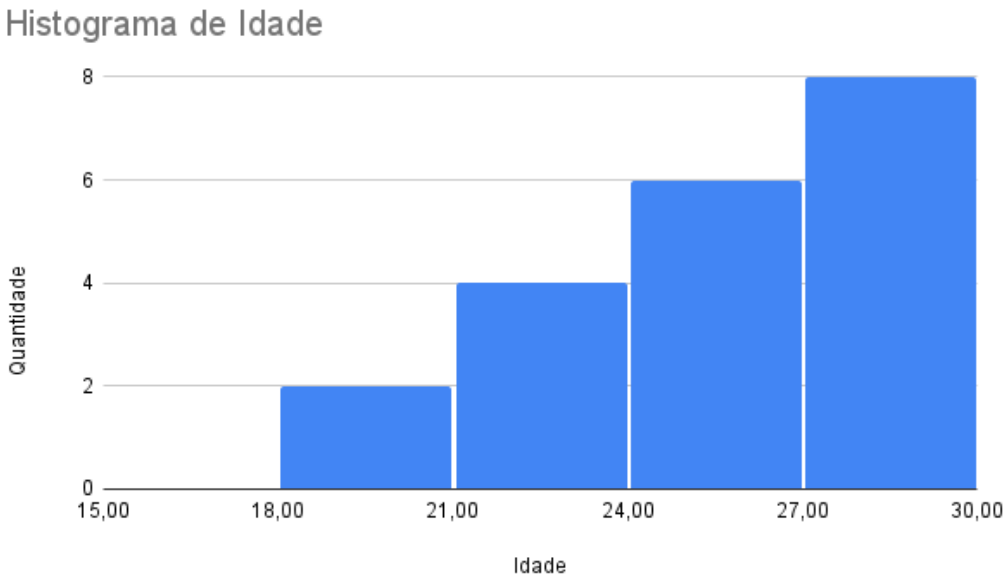


Figura C.1: *Histograma da idade dos participantes do experimento.*

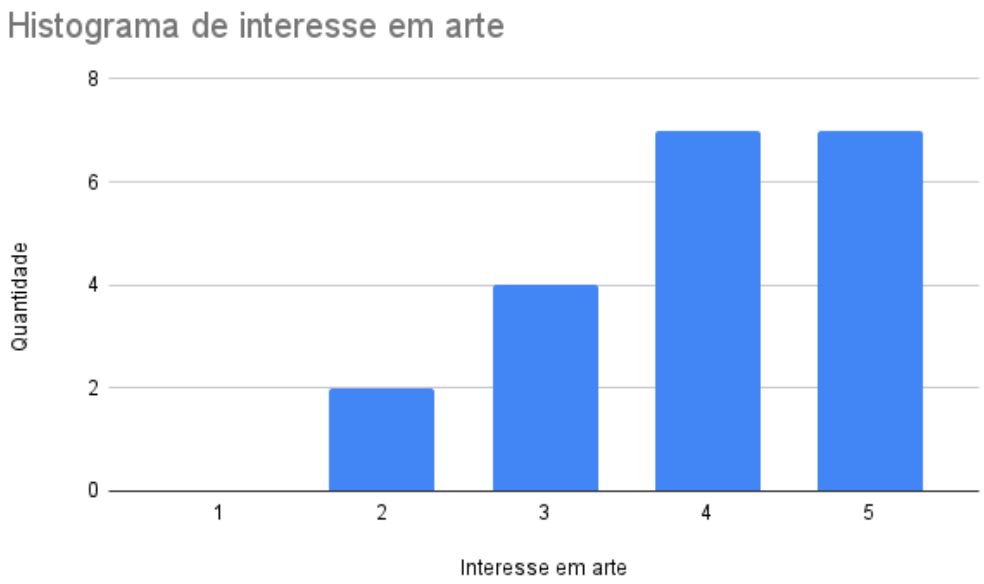


Figura C.2: *Histograma de interesse em arte dos participantes.*

Figura C.3: *Gráfico de gênero com o qual os participantes se identificam.*

Figura C.4: *Gráfico de uso semanal de redes sociais dos participantes.*

Figura C.5: *Distribuição dos participantes que moram ou não com seus pais/responsáveis.*

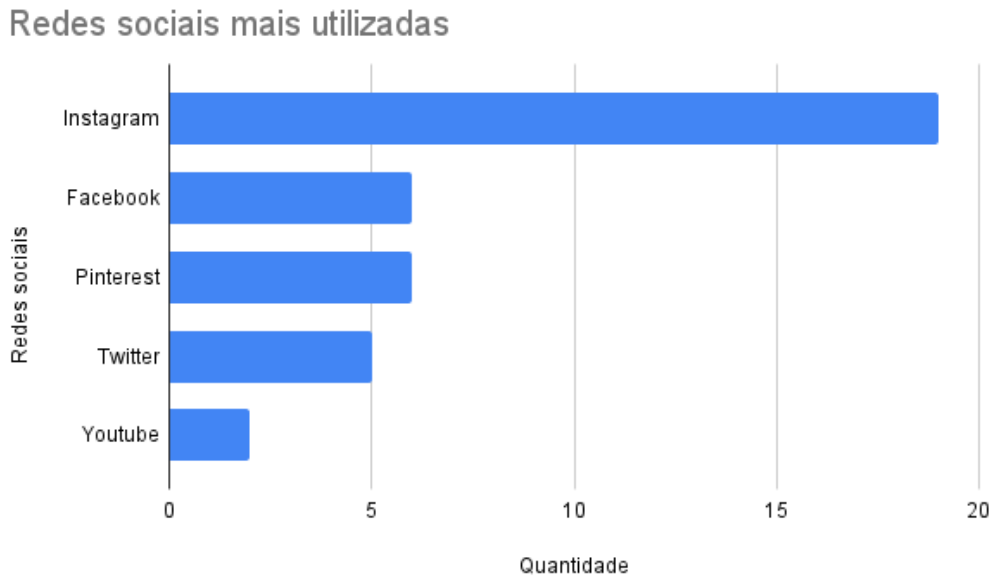


Figura C.6: *Gráfico das redes sociais mais usadas pelos participantes.*

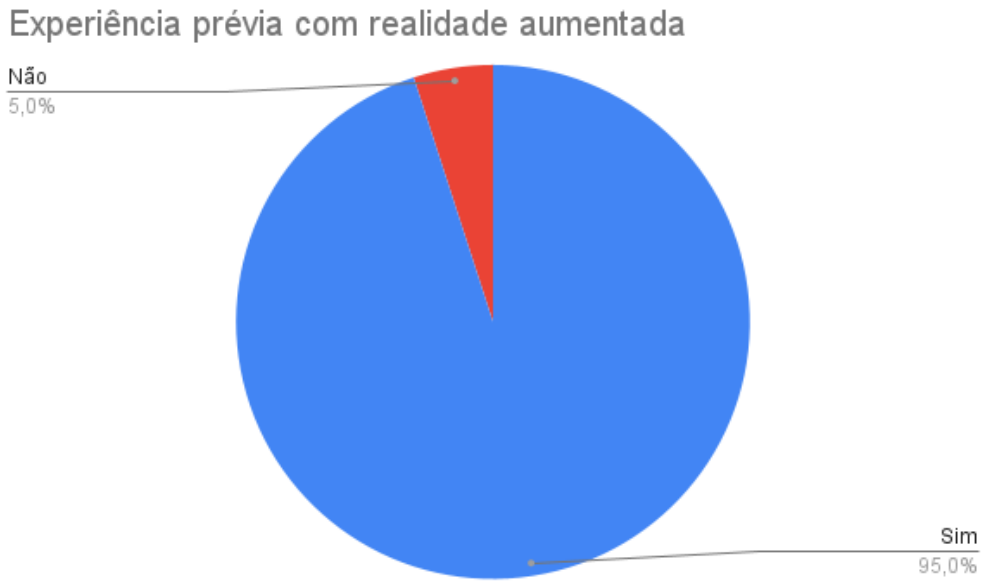


Figura C.7: *Distribuição dos participantes que já tiveram ou não experiência com realidade aumentada.*

C.2 Tempo de conclusão das tarefas

Ao longo dessa seção serão exibidos os resultados referentes ao tempo de conclusão das tarefas no primeiro, no segundo teste e também um consolidado do tempo de conclusão no experimento em geral.

C.2.1 Primeiro teste

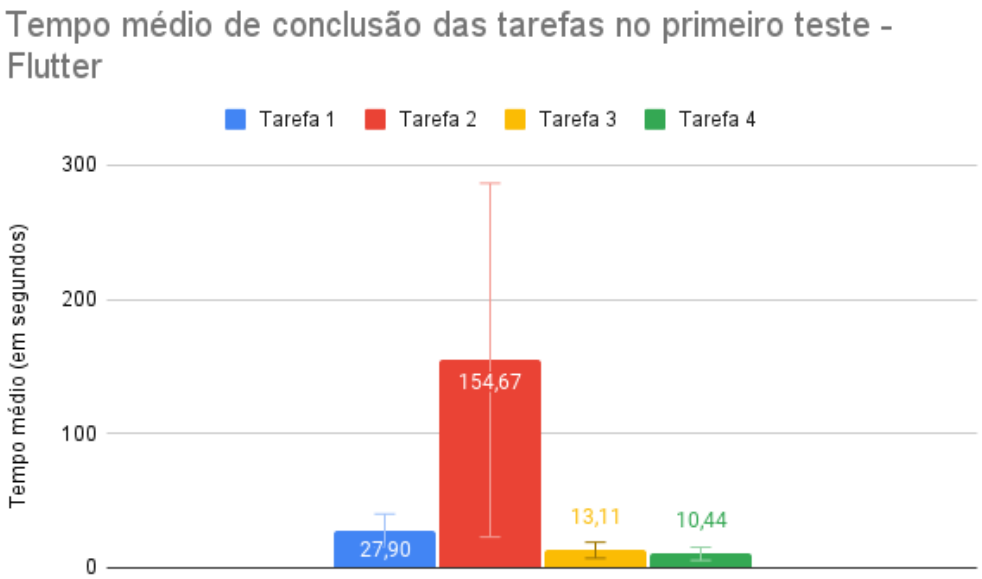


Figura C.8: *Tempo médio de conclusão das tarefas no primeiro teste realizado em Flutter.*

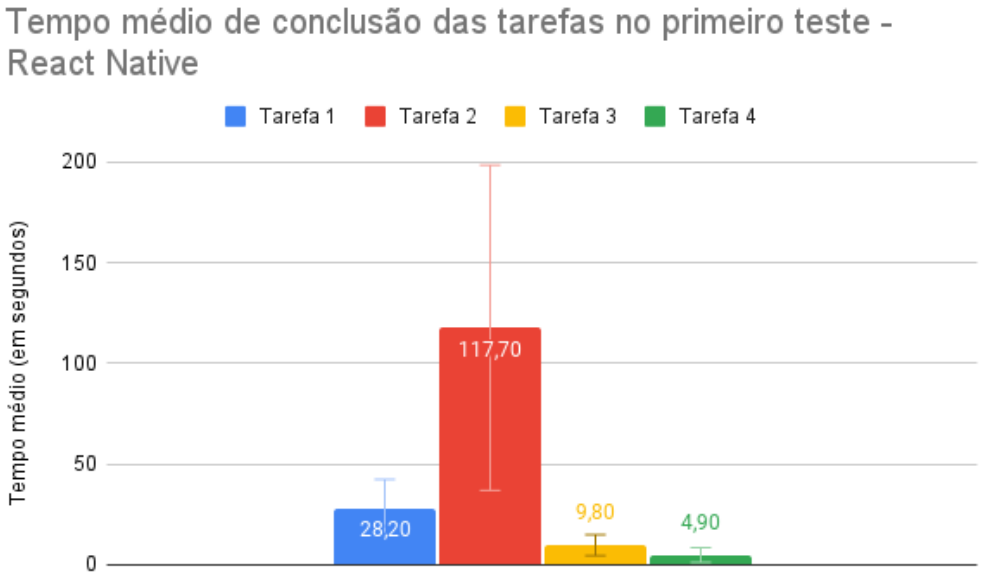


Figura C.9: *Tempo médio de conclusão das tarefas no primeiro teste realizado em React Native.*

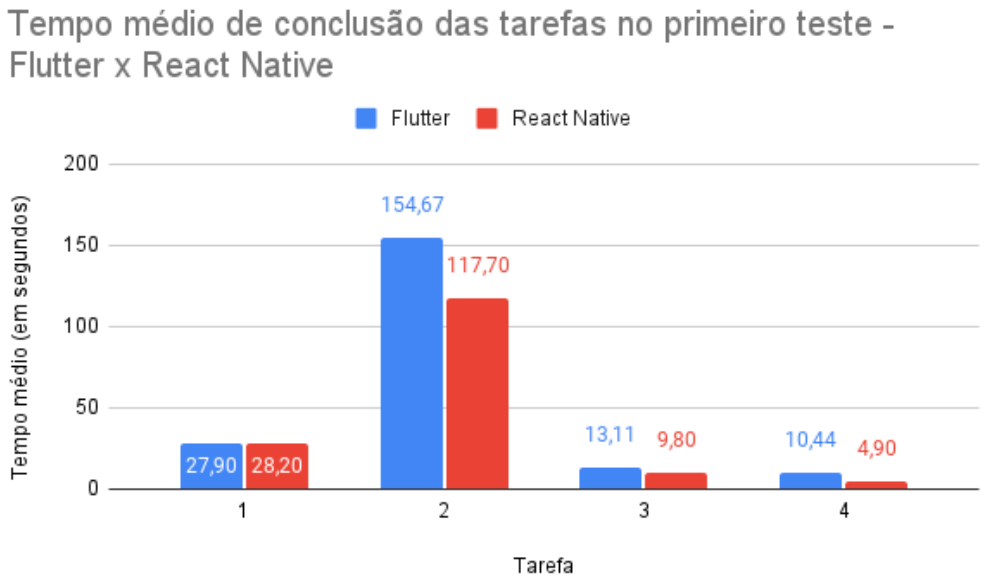


Figura C.10: *Comparativo do tempo médio de conclusão das tarefas no primeiro teste.*

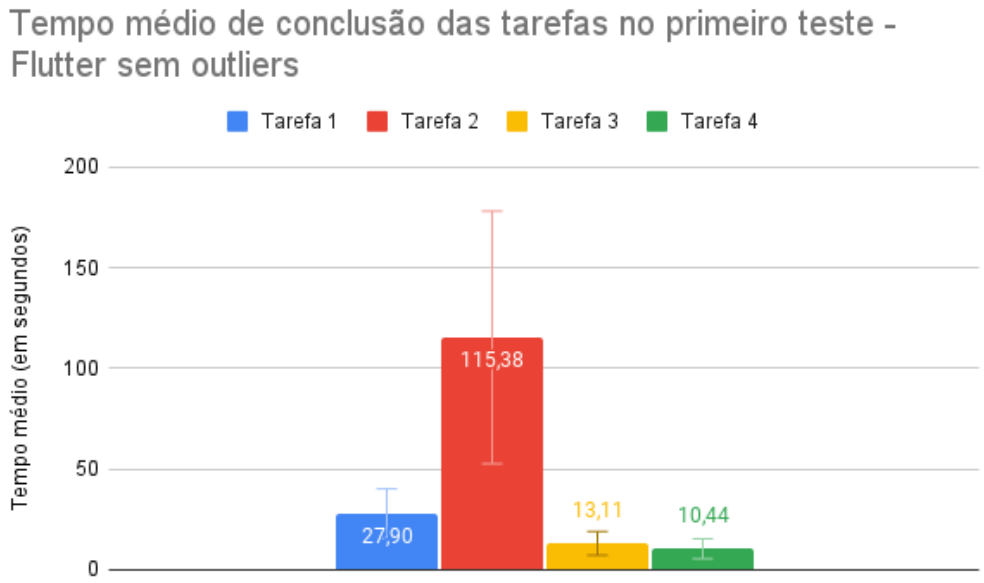


Figura C.11: *Tempo médio de conclusão das tarefas no primeiro teste realizado em Flutter desconsiderando os* outliers.

C.2.2 Segundo teste

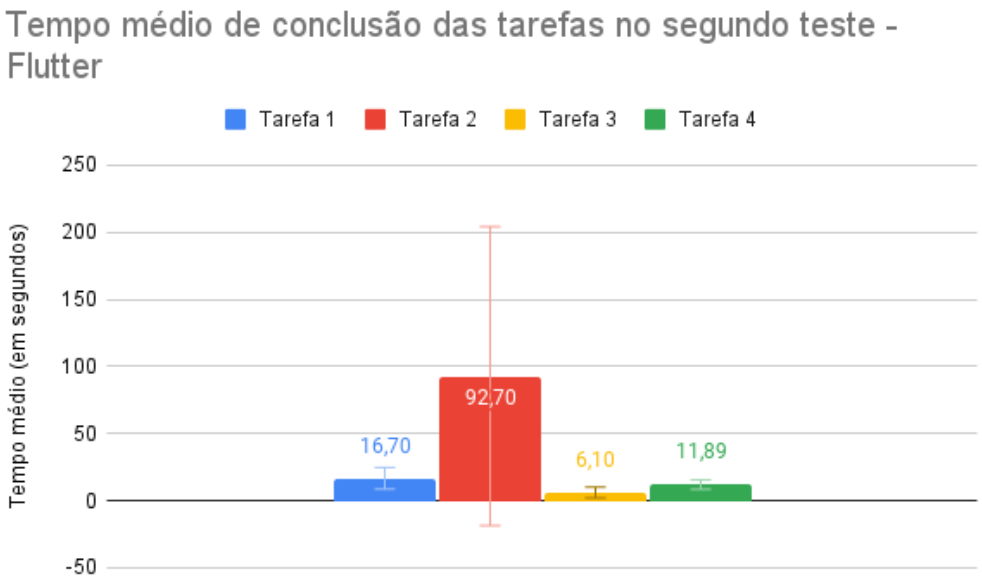


Figura C.12: *Tempo médio de conclusão das tarefas no segundo teste realizado em Flutter.*

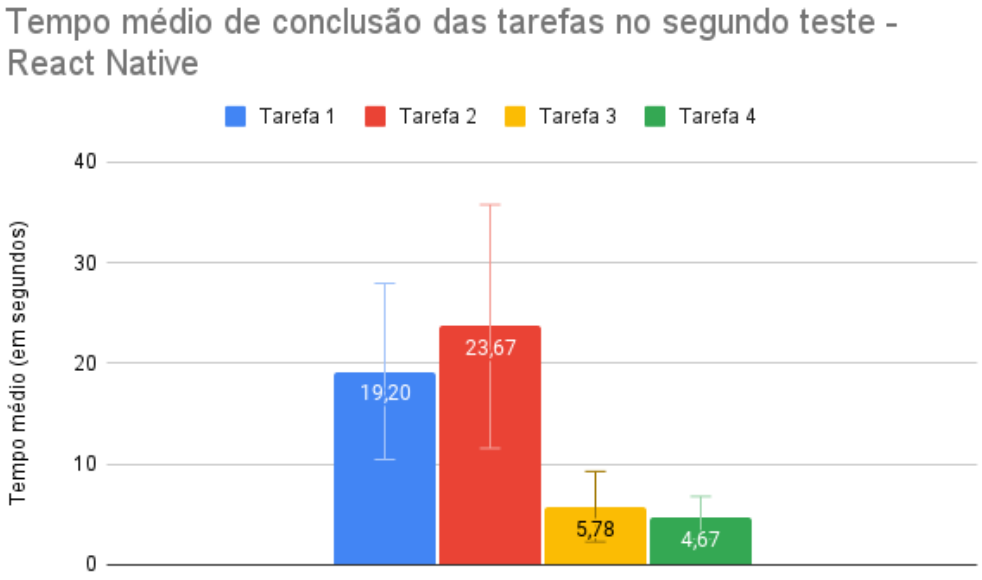


Figura C.13: *Tempo médio de conclusão das tarefas no segundo teste realizado em React Native.*

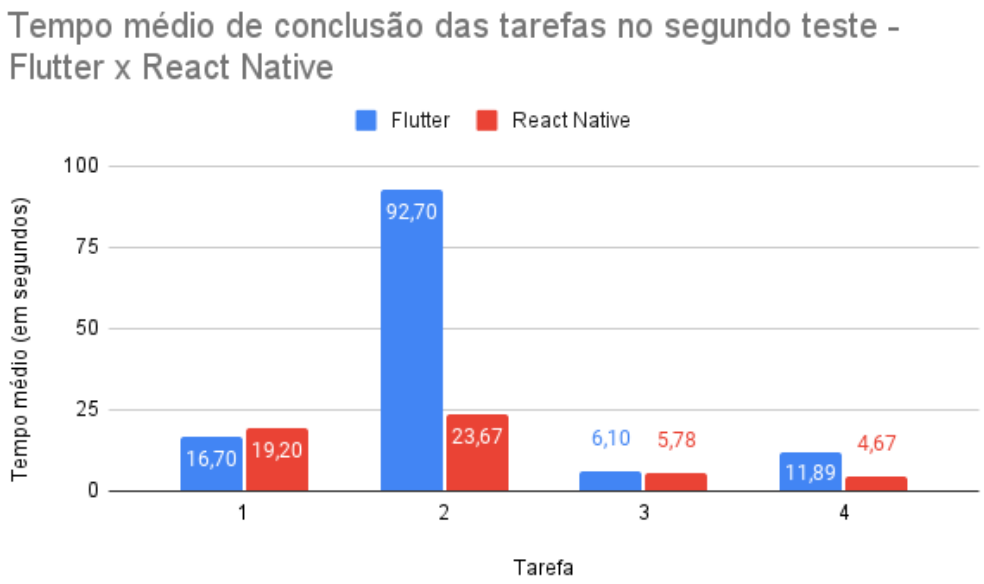


Figura C.14: *Comparativo do tempo médio de conclusão das tarefas no segundo teste.*

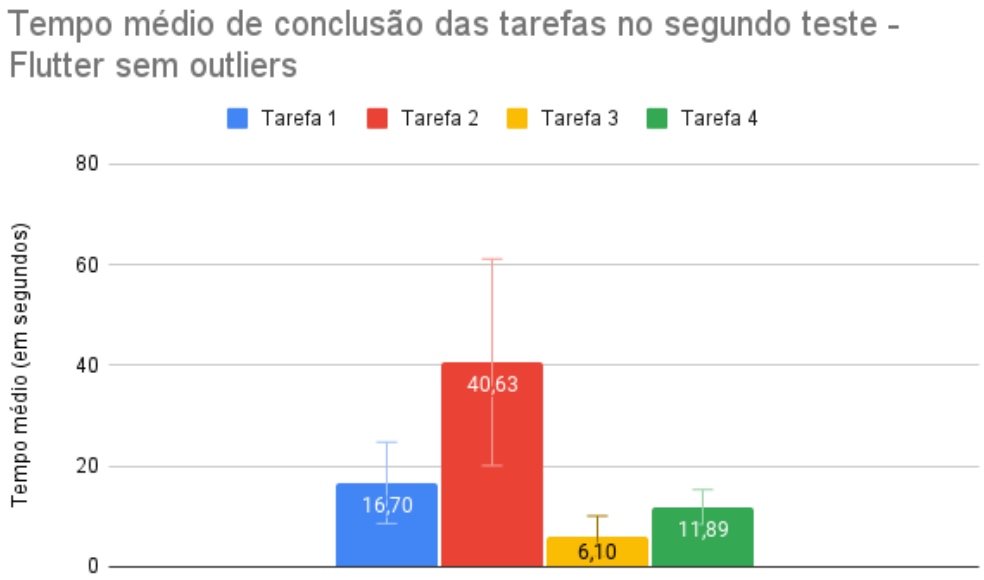


Figura C.15: *Tempo médio de conclusão das tarefas no segundo teste realizado em Flutter desconsiderando os* outliers.

C.2.3 Experimento geral

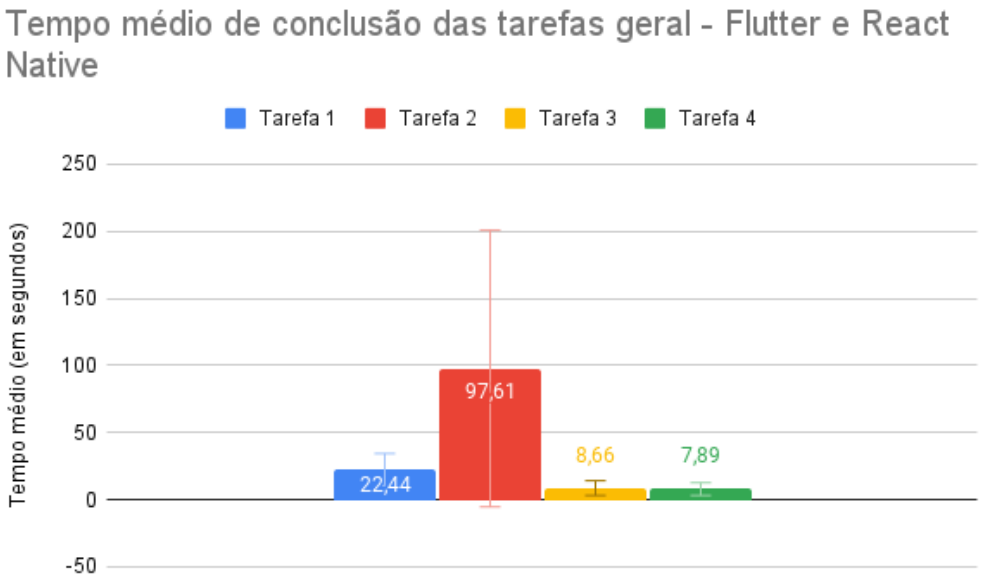


Figura C.16: *Tempo médio de conclusão das tarefas em geral, considerando os dados dos testes em React Native e em Flutter.*

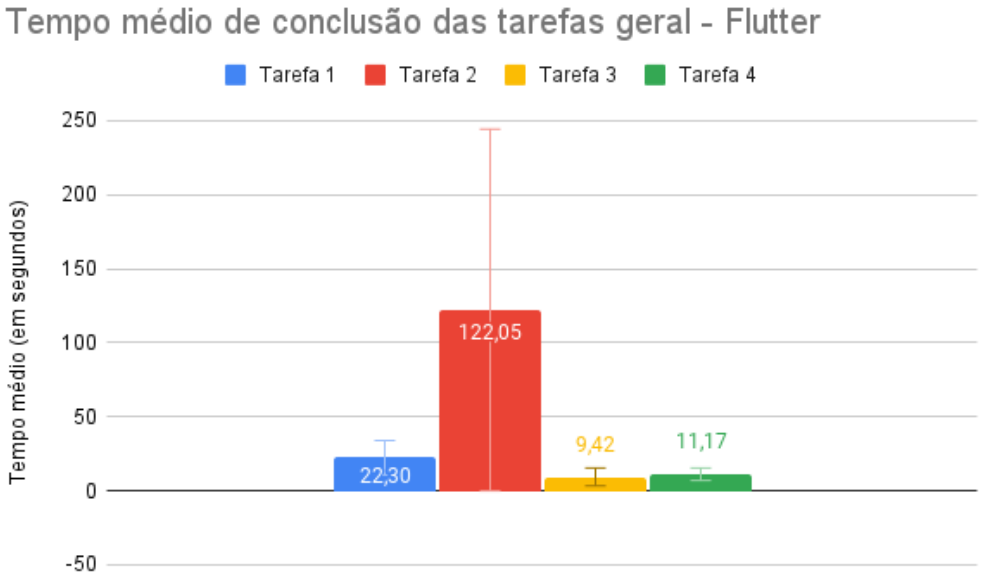


Figura C.17: *Tempo médio de conclusão das tarefas em geral realizado em Flutter.*

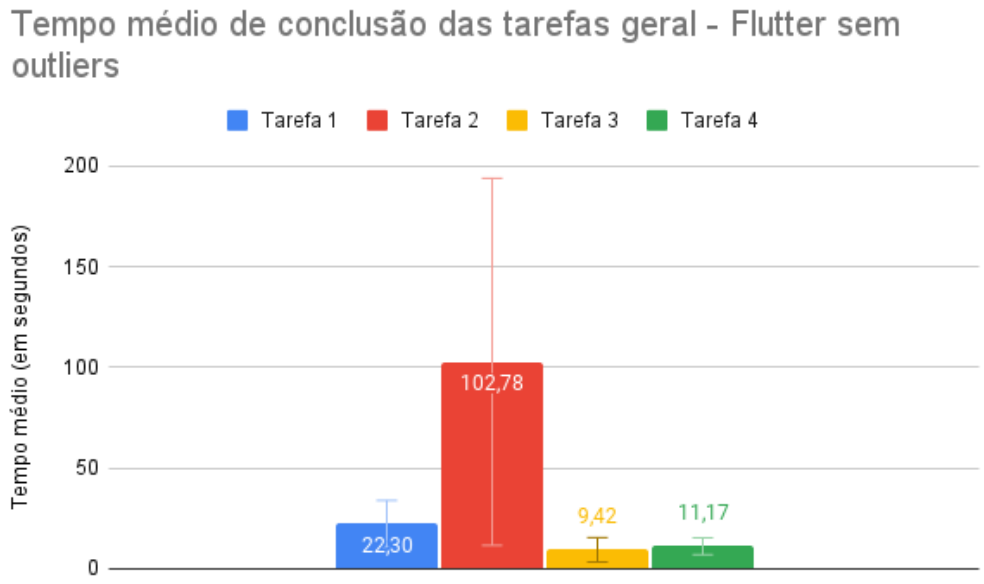


Figura C.18: *Tempo médio de conclusão das tarefas em geral realizado em Flutter desconsiderando os* outliers.

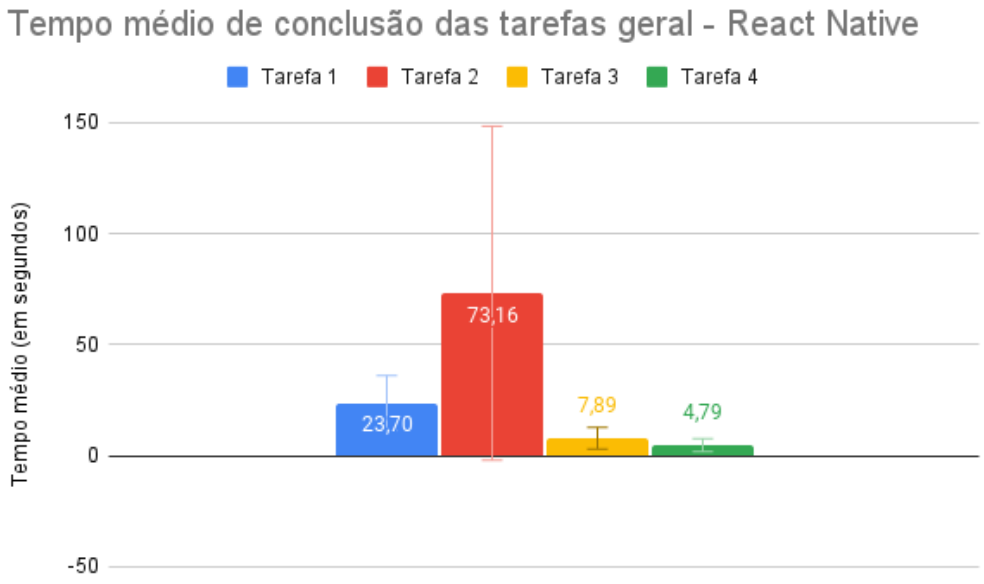


Figura C.19: *Tempo médio de conclusão das tarefas geral realizado em React Native.*

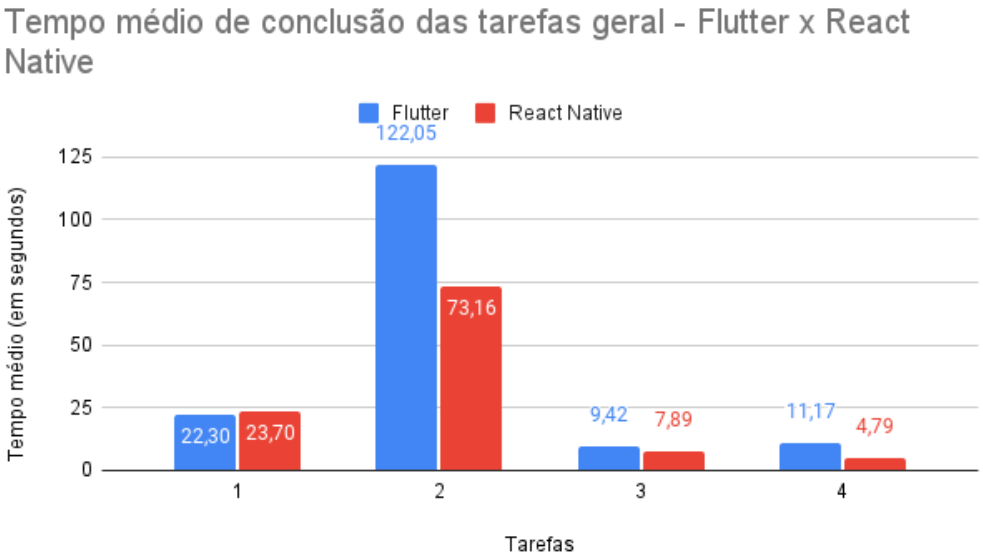


Figura C.20: *Comparativo do tempo médio de conclusão das tarefas em geral.*

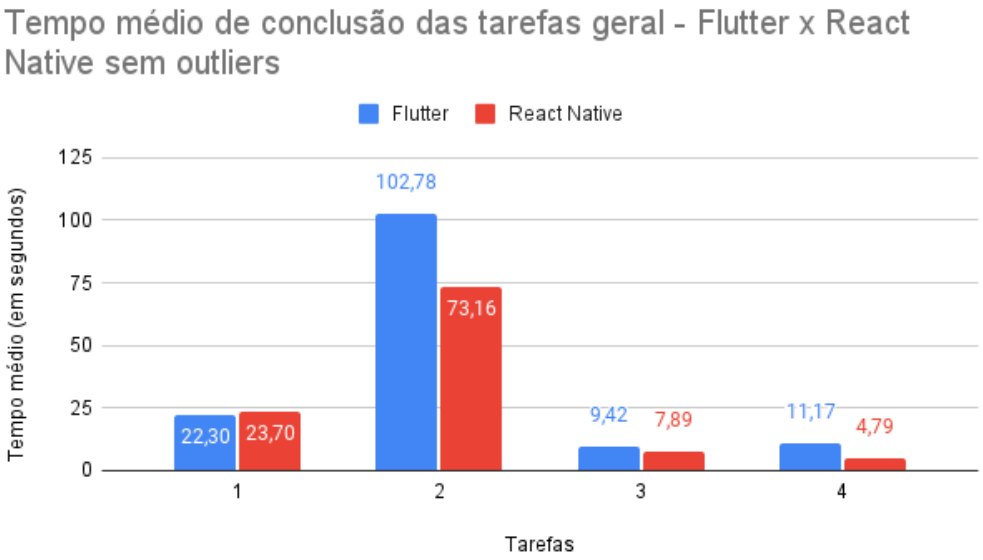


Figura C.21: *Comparativo do tempo médio de conclusão das tarefas em geral desconsiderando os* outliers.

C.3 Dificuldade de conclusão das tarefas

Assim como na seção anterior, vamos analisar os dados resultados obtidos no primeiro teste, depois no segundo e por último os resultados considerando todos os dados coletados, só que dessa vez em relação à dificuldade percebida pelos participantes para conclusão das tarefas.

C.3.1 Primeiro teste

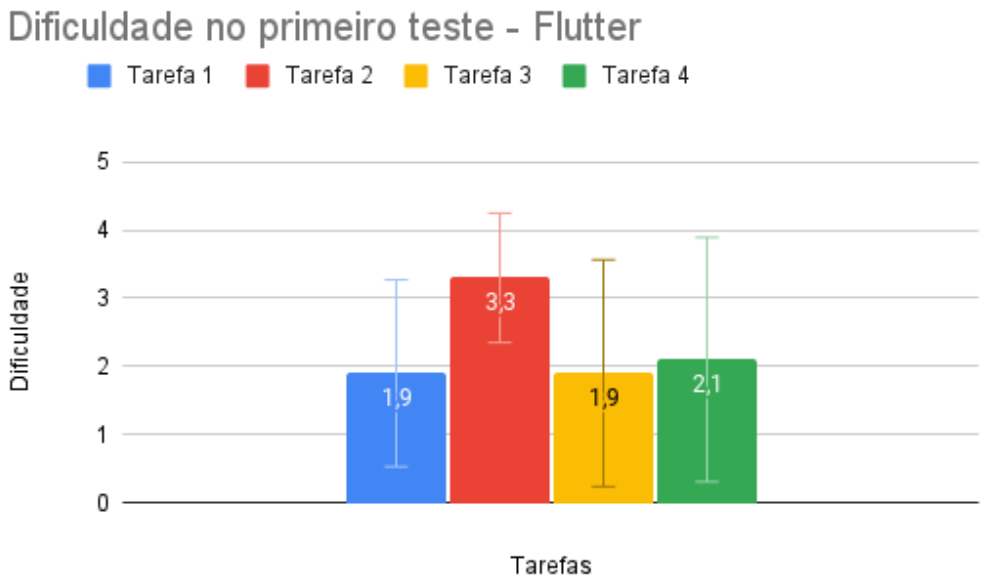


Figura C.22: *Dificuldade percebida no primeiro teste em Flutter.*

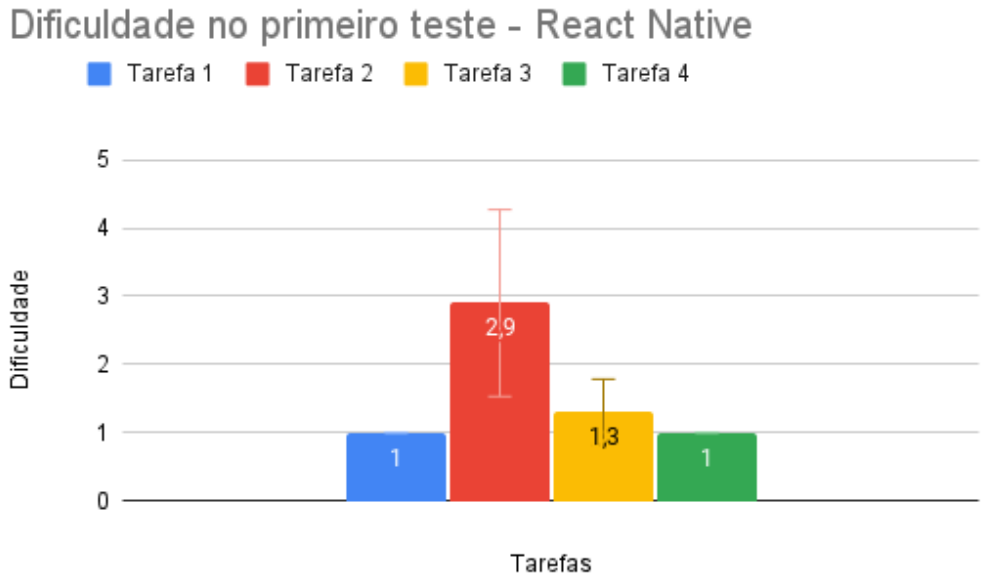


Figura C.23: *Dificuldade percebida no primeiro teste em React Native.*

C.3.2 Segundo teste

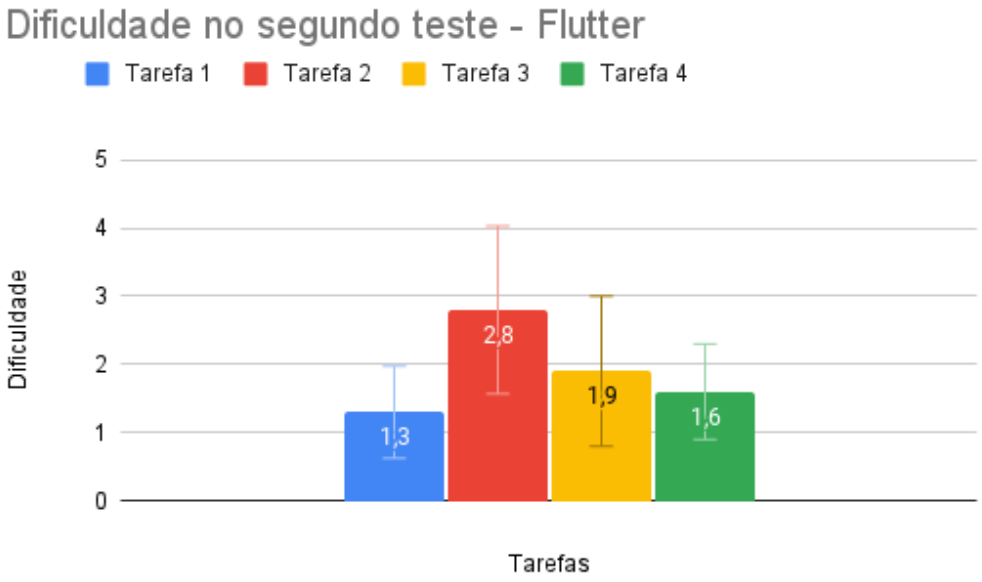


Figura C.24: *Dificuldade percebida no segundo teste em Flutter.*

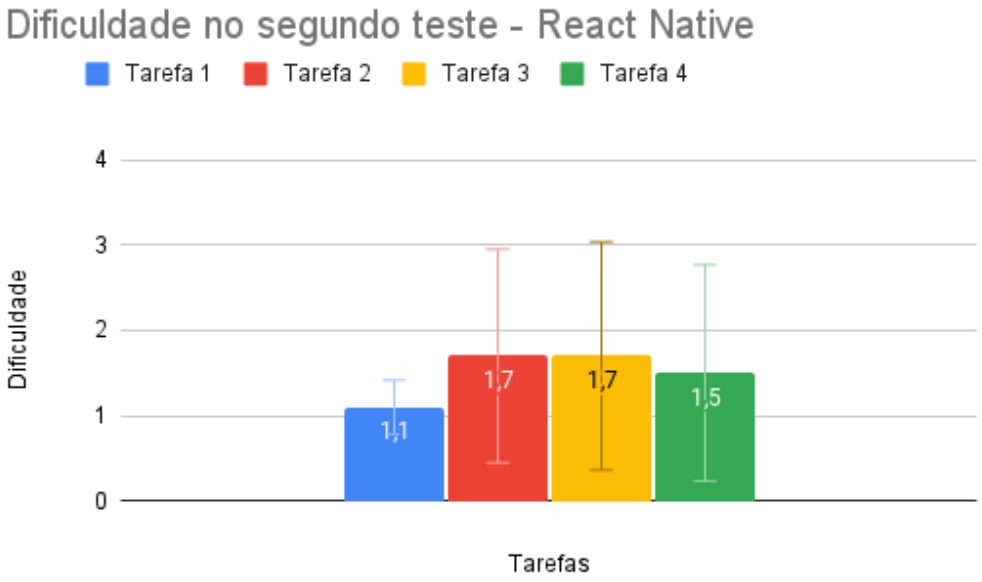


Figura C.25: *Dificuldade percebida no segundo teste em React Native.*

C.3.3 Experimento geral

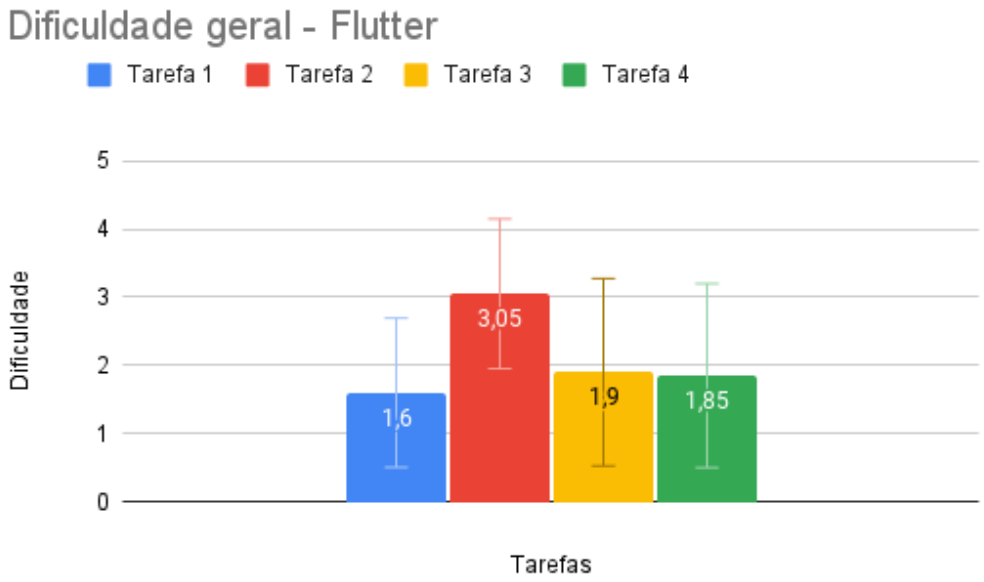


Figura C.26: *Dificuldade geral percebida em Flutter.*

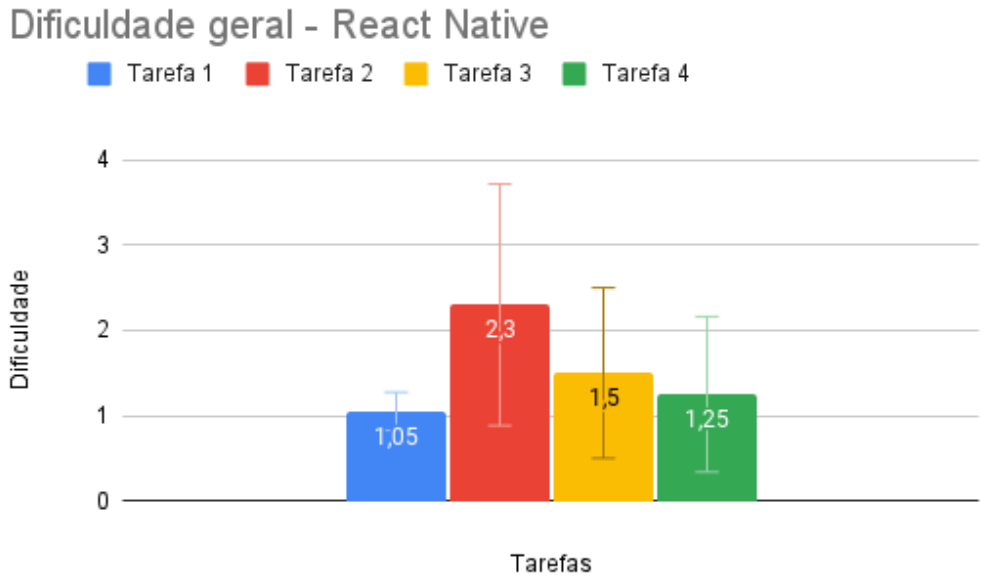


Figura C.27: *Dificuldade geral percebida em React Native.*

C.4 Experiência do usuário

Nessa seção serão apresentados os gráficos referentes as análises de experiência de usuário resultantes do UEQ aplicado nesse experimento.

C.4.1 Primeiro teste

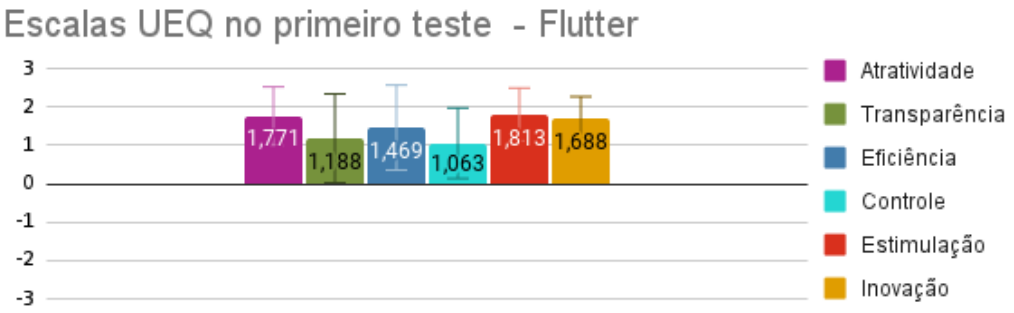


Figura C.28: *Escalas UEQ resultantes do primeiro teste em Flutter.*

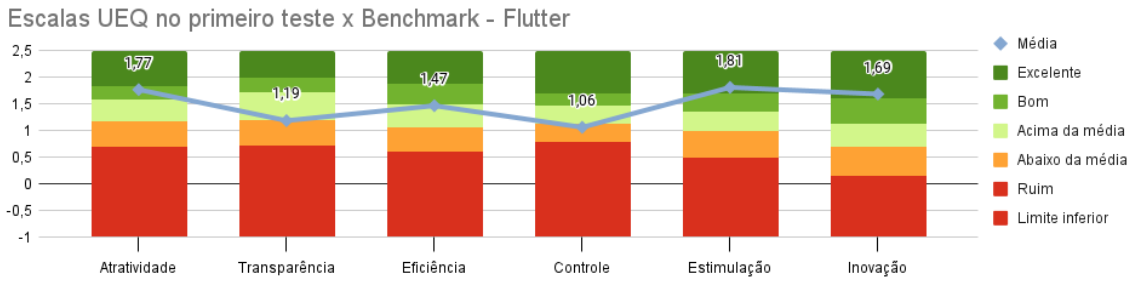


Figura C.29: *Escalas UEQ resultantes do primeiro teste em Flutter comparadas com valores de referência.*

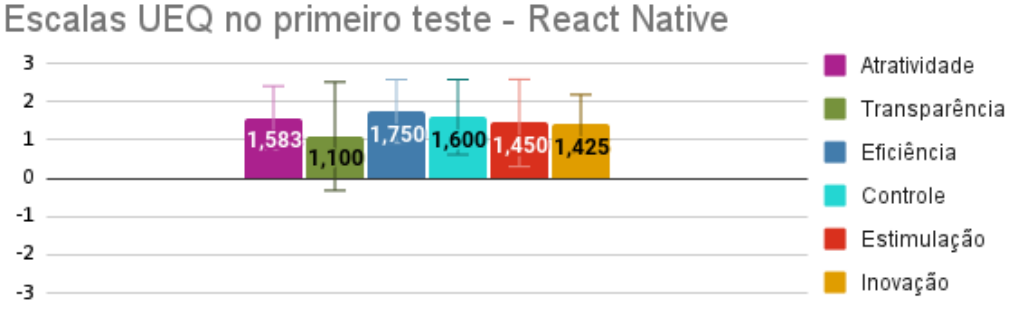


Figura C.30: *Escalas UEQ resultantes do primeiro teste em React Native.*

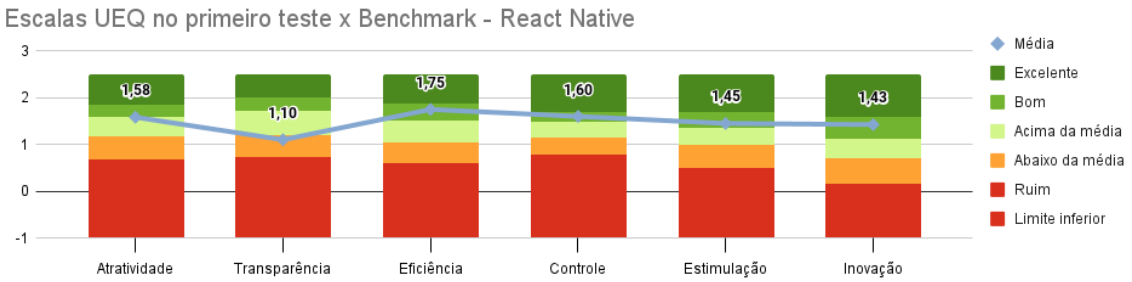


Figura C.31: *Escalas UEQ resultantes do primeiro teste em React Native comparadas com valores de referência.*

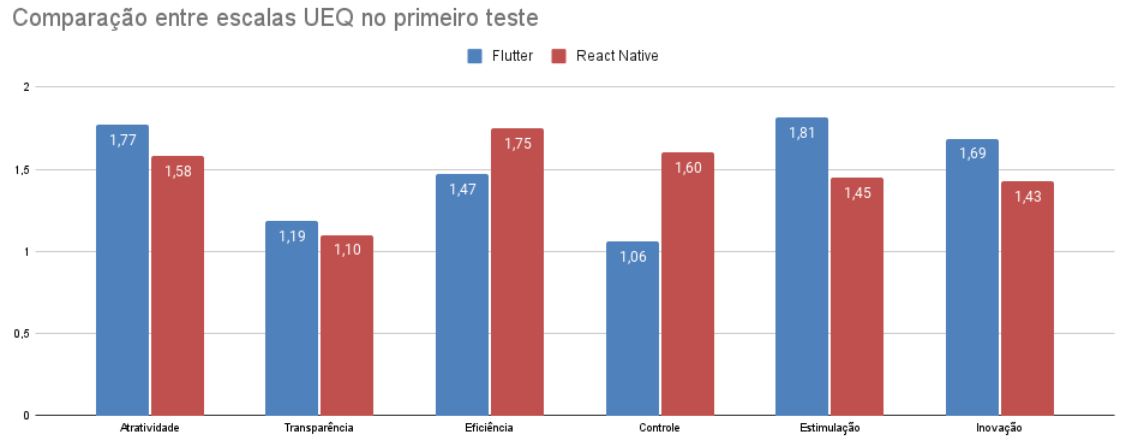


Figura C.32: *Comparativo entre as escalas UEQ resultantes do primeiro teste.*

C.4.2 Segundo teste

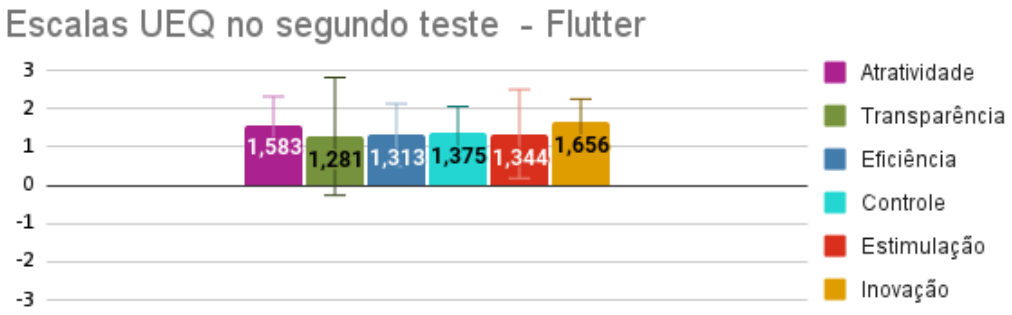


Figura C.33: *Escalas UEQ resultantes do segundo teste em Flutter.*

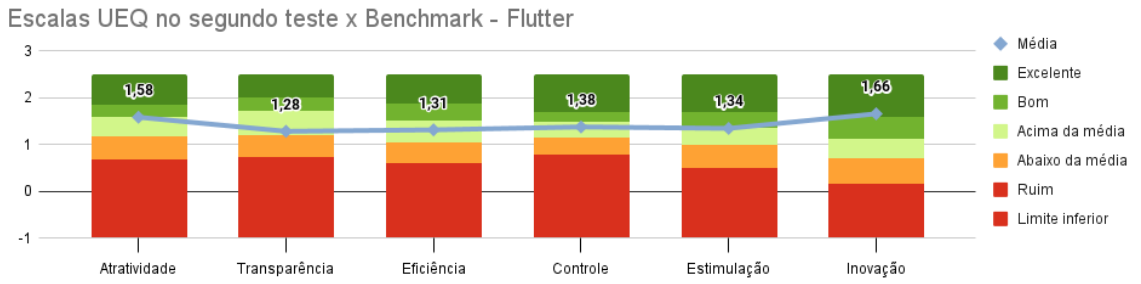


Figura C.34: *Escalas UEQ resultantes do segundo teste em Flutter comparadas com valores de referência.*

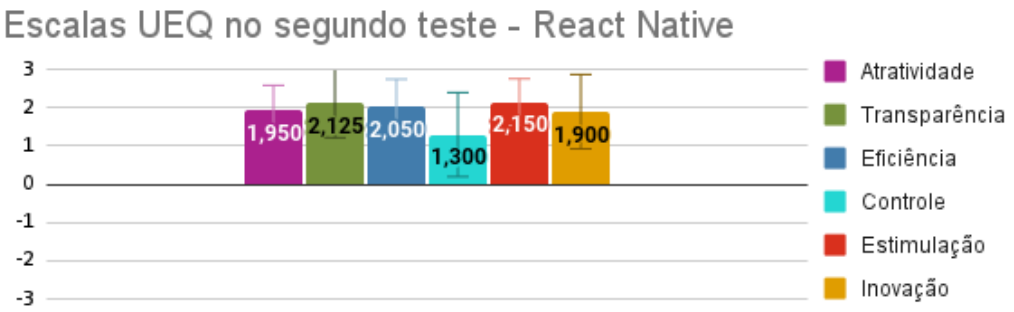


Figura C.35: *Escalas UEQ resultantes do segundo teste em React Native.*

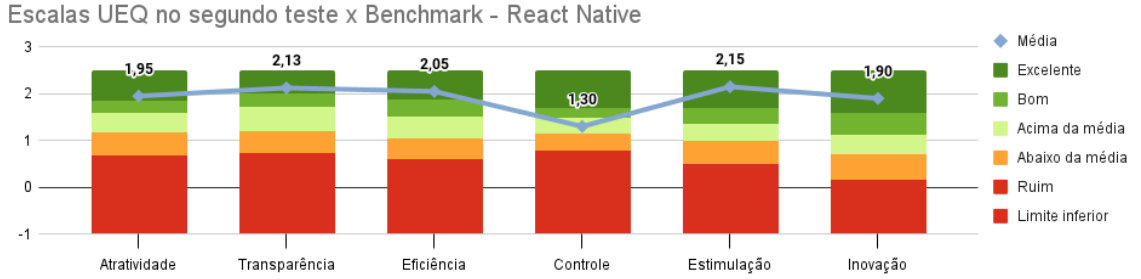


Figura C.36: *Escalas UEQ resultantes do segundo teste em React Native comparadas com valores de referência.*

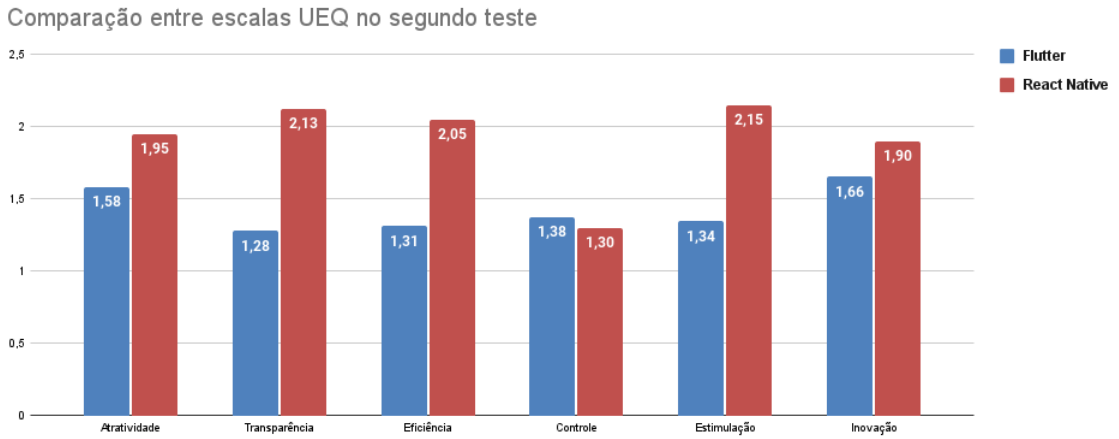


Figura C.37: *Comparativo entre as escalas UEQ resultantes do segundo teste.*

C.4.3 Experimento geral

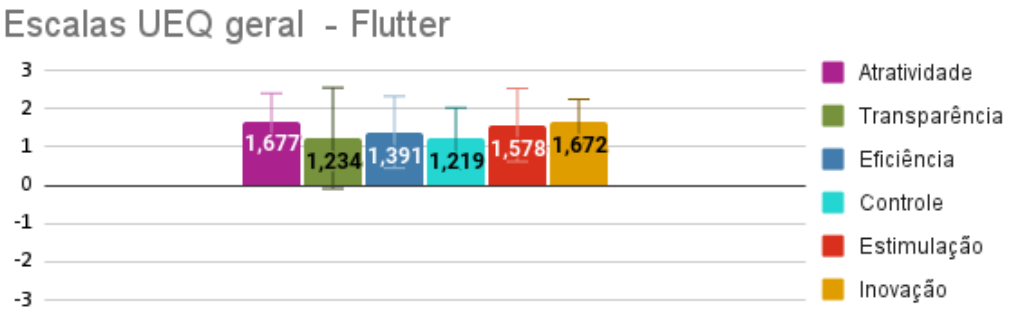


Figura C.38: *Escalas UEQ resultantes do experimento todo aplicado em Flutter.*

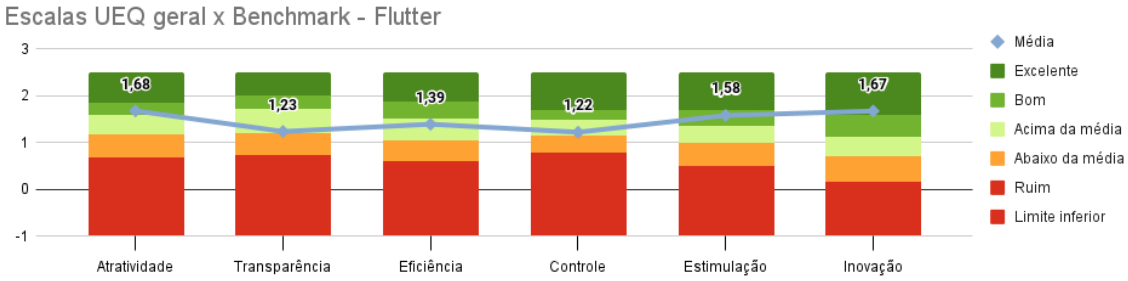


Figura C.39: *Escalas UEQ resultantes do experimento todo aplicado em Flutter comparadas com valores de referência.*

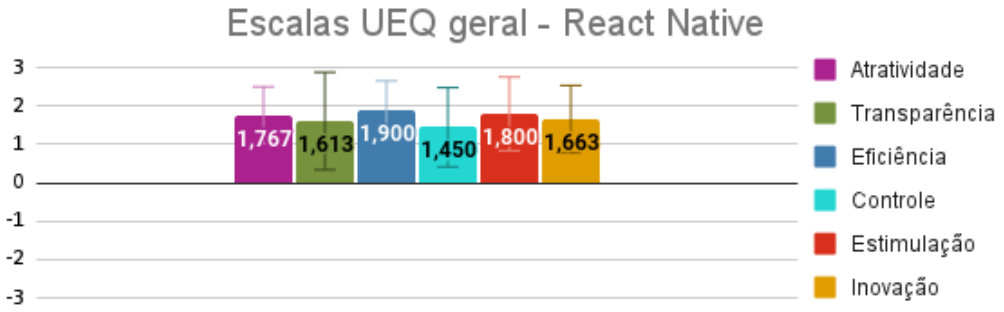


Figura C.40: *Escalas UEQ resultantes do experimento todo aplicado em React Native.*

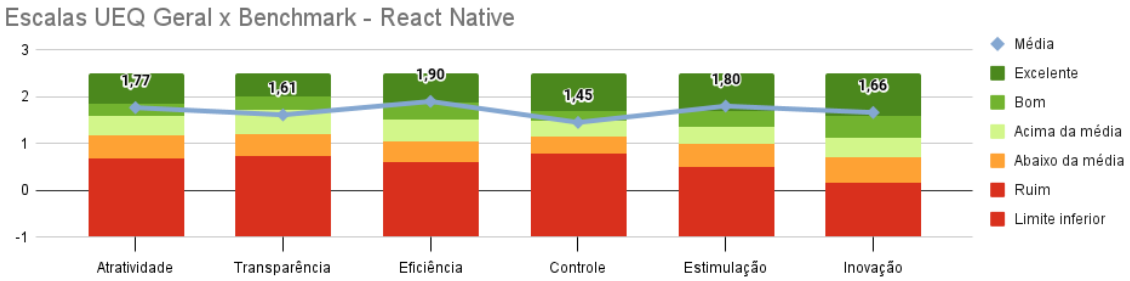


Figura C.41: *Escalas UEQ resultantes do experimento todo aplicado em React Native comparadas com valores de referência.*

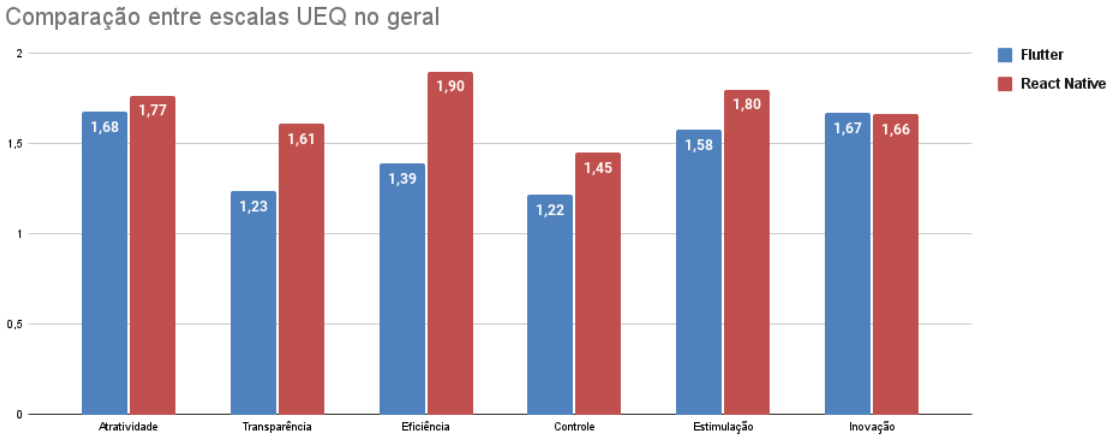


Figura C.42: *Comparativo entre as escalas UEQ resultantes do experimento.*

Referências

[Azuma 1997] Ronald T. Azuma. "A survey of augmented reality". Em: *Presence: Teleoperators and Virtual Environments* 6.4 (ago. de 1997), pgs. 355–385. DOI: 10.1162/pres.1997.6.4.355 (citado na pg. 3).

[Barquet *et al.* 2011] Ana Barquet, Vitor Cunha, Maicon Oliveira e Henrique Rozenfeld. "Business model elements for product-service system". Em: mar. de 2011, pgs. 332–337. ISBN: 978-3-642-19688-1. DOI: 10.1007/978-3-642-19689-8_58 (citado na pg. 4).

[Brown 2010] D.M. Brown. *Communicating Design: Developing Web Site Documentation for Design and Planning*. Voices That Matter. Pearson Education, 2010. ISBN: 9780131385412. URL: https://books.google.com.br/books?id=6dlFc4iwbuIC (citado na pg. 4).

[Durrant-Whyte e Bailey 2006] H. Durrant-Whyte e T. Bailey. "Simultaneous localization and mapping: part i". Em: *IEEE Robotics Automation Magazine* 13.2 (2006), pgs. 99–110. DOI: 10.1109/MRA.2006.1638022 (citado nas pgs. viii, 18, 19, 21).

[Emmert-Streib e Dehmer 2019] Frank Emmert-Streib e Matthias Dehmer. "Understanding statistical hypothesis testing: the logic of statistical inference". Em: *Machine Learning and Knowledge Extraction* 1 (ago. de 2019), pgs. 945–961. DOI: 10.3390/make1030054 (citado na pg. 41).

[Flutter 2021] Flutter. *Flutter architectural overview*. URL: https://flutter.dev/docs/resources/architectural-overview (acesso em 05/09/2021) (citado na pg. 10).

[Gitlab 2021] Gitlab. *Is it any good?* URL: https://about.gitlab.com/is-it-any-good/#gitlab-is-the-most-used-devops-tool (acesso em 15/08/2021) (citado na pg. 49).

[Google 2021a] Google. *Meet Android Studio*. Set. de 2021. URL: https://developer.android.com/studio/intro (acesso em 05/09/2021) (citado na pg. 50).

[Google 2021b] Google. *Choose your development environment*. URL: https://developers.google.com/ar/develop (acesso em 13/10/2021) (citado na pg. 17).

[Google 2021c] Google. *Fundamental concepts*. URL: https://developers.google.com/ar/develop/fundamentals (acesso em 13/10/2021) (citado nas pgs. 18, 22).

[Han *et al.* 2018] Songqi Han, Weibo Yu, Hongtao Yang e Shicheng Wan. "An improved corner detection algorithm based on harris". Em: *2018 Chinese Automation Congress (CAC)*. 2018, pgs. 1575–1580. DOI: 10.1109/CAC.2018.8623814 (citado na pg. 24).

[Harris e Stephens 1988] Chris Harris e Mike Stephens. "A combined corner and edge detector". Em: *In Proc. of Fourth Alvey Vision Conference*. 1988, pgs. 147–151 (citado na pg. 23).

[Institute 2013] Project Management Institute. *A Guide to the Project Management Body of Knowledge: PMBOK Guide*. PMBOK® Guide Series. Project Management Institute, 2013. ISBN: 9781935589679. URL: https://books.google.com.br/books?id=FpatMQEACAAJ (citado na pg. 4).

[Jabbari *et al.* 2016] Ramtin Jabbari, Nauman Ali, Kai Petersen e Binish Tanveer. "What is devops?: a systematic mapping study on definitions and practices". Em: mai. de 2016, pgs. 1–11. DOI: 10.1145/2962695.2962707 (citado na pg. 49).

[Kaufmann 2003] Hannes Kaufmann. "Collaborative augmented reality in education". Em: *Institute of Software Technology and Interactive Systems, Vienna University of Technology* (2003) (citado na pg. 1).

[Laugwitz *et al.* 2008] Bettina Laugwitz, Theo Held e Martin Schrepp. "Construction and evaluation of a user experience questionnaire". Em: vol. 5298. Nov. de 2008, pgs. 63–76. ISBN: 978-3-540-89349-3. DOI: 10.1007/978-3-540-89350-9_6 (citado na pg. 33).

[Liu *et al.* 2018] Haomin Liu, Mingyu Chen, Guofeng Zhang, Hujun Bao e Yingze Bao. "Ice-ba: incremental, consistent and efficient bundle adjustment for visual-inertial slam". Em: jun. de 2018, pgs. 1974–1982. DOI: 10.1109/CVPR.2018.00211 (citado na pg. 22).

[Lowe 2004] David G Lowe. "Distinctive image features from scale-invariant keypoints". Em: *International journal of computer vision* 60.2 (2004), pgs. 91–110 (citado na pg. 24).

[MacKenzie 2013] I. Scott MacKenzie. *Human-Computer Interaction: An Empirical Research Perspective*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN: 0124058655 (citado na pg. 31).

[Meirelles *et al.* 2019] Paulo Meirelles, Carla S. R. Aguiar, Felipe Assis, Rodrigo Siqueira e Alfredo Goldman. "A students' perspective of native and cross-platform approaches for mobile application development". Em: *Computational Science and Its Applications – ICCSA 2019*. Ed. por Sanjay Misra *et al.* Cham: Springer International Publishing, 2019, pgs. 586–601. ISBN: 978-3-030-24308-1 (citado na pg. 1).

[Menditto *et al.* 2007] Antonio Menditto, Marina Patriarca e Bertil Magnusson. "Understanding the meaning of accuracy, trueness and precision". Em: *Accreditation and quality assurance* 12.1 (2007), pgs. 45–47 (citado na pg. 41).

[Meyerovich e Rabkin 2013] Leo A. Meyerovich e Ariel S. Rabkin. "Empirical analysis of programming language adoption". Em: *SIGPLAN Not.* 48.10 (out. de 2013), pgs. 1–18. ISSN: 0362-1340. DOI: 10.1145/2544173.2509515. URL: https://doi.org/10.1145/2544173.2509515 (citado na pg. 15).

[Montemerlo *et al.* 2002] Michael Montemerlo, Sebastian Thrun, Daphne Koller e Ben Wegbreit. "Fastslam: a factored solution to the simultaneous localization and mapping problem". Em: nov. de 2002 (citado nas pgs. 21, 22).

[Nowacki e Woda 2020] Paweł Nowacki e Marek Woda. "Capabilities of arcore and arkit platforms for ar/vr applications". Em: jan. de 2020, pgs. 358–370. ISBN: 978-3-030-15405-9. DOI: 10.1007/978-3-030-19501-4_36 (citado na pg. 26).

[Overflow 2021] Stack Overflow. *2021 Developer Survey*. Mai. de 2021. URL: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies (acesso em 16/12/2021) (citado na pg. 2).

[Research 2021] Great View Research. *Augmented Reality Market Size, Share & Trends Analysis Report By Component, By Display (HMD & Smart Glass, HUD, Handheld Devices), By Application, By Region, And Segment Forecasts, 2021 - 2028*. Fev. de 2021. URL: https://www.grandviewresearch.com/industry-analysis/augmented-reality-market (acesso em 16/12/2021) (citado na pg. 1).

[M. Ribeiro e I. Ribeiro 2004] Maria Ribeiro e Isabel Ribeiro. *Kalman and Extended Kalman Filters: Concept, Derivation and Properties*. Rel. técn. Abr. de 2004 (citado na pg. 20).

[Ries 2011] E. Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business, 2011. ISBN: 9780307887894. URL: https://books.google.com.br/books?id=r9x-OXdzpPcC (citado na pg. 4).

[Sánchez *et al.* 2018] Javier Sánchez, Nelson Monzón e Agustín Salgado. "An analysis and implementation of the harris corner detector". Em: *Image Processing On Line* 8 (out. de 2018), pgs. 305–328. DOI: 10.5201/ipol.2018.229 (citado na pg. 23).

[Se *et al.* 2001] S. Se, D. Lowe e J. Little. "Vision-based mobile robot localization and mapping using scale-invariant features". Em: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, 2051–2058 vol.2. DOI: 10.1109/ROBOT.2001.932909 (citado na pg. 26).

[Simon 2001] Dan Simon. "Kalman filtering". Em: *Embedded systems programming* 14.6 (2001), pgs. 72–79 (citado na pg. 20).

[*ISO/IEC 25010:2011* 2011] *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Standard ISO/IEC 25010:2011. Geneva, Switzerland: International Organization for Standardization, 2011. URL: https://www.iso.org/standard/35733.html (citado na pg. 14).

[Thrun *et al.* 2005] Sebastian Thrun, Wolfram Burgard e Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623 (citado na pg. 21).

[Tukey *et al.* 1977] John W Tukey *et al. Exploratory data analysis*. Vol. 2. Reading, Mass., 1977 (citado na pg. 34).

[Vilček e Jakopec 2017] Tena Vilček e Tomislav Jakopec. "Comparative analysis of tools for development of native and hybrid mobile applications". Em: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2017, pgs. 1516–1521. DOI: 10.23919/MIPRO.2017.7973662 (citado na pg. 1).

[Vithani e Kumar 2014] Tejas Vithani e Anand Kumar. "Modeling the mobile application development lifecycle". Em: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 1. 2014, pgs. 596–600 (citado na pg. 49).

[Weihrich 1982] Heinz Weihrich. "The tows matrix—a tool for situational analysis". Em: *Long Range Planning* 15.2 (1982), pgs. 54–66. ISSN: 0024-6301. DOI: https://doi.org/10.1016/0024-6301(82)90120-0. URL: https://www.sciencedirect.com/science/article/pii/0024630182901200 (citado na pg. 4).

[Yao *et al.* 2014] Lin Yao *et al.* "Using physiological measures to evaluate user experience of mobile applications". Em: *Engineering Psychology and Cognitive Ergonomics*. Ed. por Don Harris. Cham: Springer International Publishing, 2014, pgs. 301–310. ISBN: 978-3-319-07515-0 (citado na pg. 31).