UNIVERSIDADE FEEVALE

LUIZ EDUARDO S. VIEIRALVES

ANÁLISE DE LINGUAGEM MULTIPLATAFORMA COM FOCO EM FLUTTER

Novo Hamburgo

2020

LUIZ EDUARDO S. VIEIRALVES

ANÁLISE DE LINGUAGEM MULTIPLATAFORMA COM FOCO EM FLUTTER

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Prof. Paulo Ricardo Muniz Barros

Novo Hamburgo

2020

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

Aos meus pais, Drival e Osmana por sempre priorizarem meus estudos e incentivarem minha dedicação.

A minha Esposa Sandra, pela compreensão e auxílio durante todo o desenvolvimento deste trabalho.

Ao professor Paulo Ricardo, pela disposição e pelas sugestões concedidas durante a construção deste trabalho.

A todos que, de alguma forma, contribuíram para a conclusão deste trabalho.

Muito obrigado!

RESUMO

Com o crescimento da quantidade de sistemas operacionais, assim como a crescente variação de versões deles, é possível perceber um aumento na dificuldade de manutenção de programas existentes, assim como na criação de novos programas. Essa dificuldade se apresenta tanto para sistemas operacionais diferenciados quanto para versões diferentes de um sistema operacional. Considerando essa dificuldade, muitas maneiras têm sido apresentadas para realizar o desenvolvimento multiplataforma. *Frameworks* têm sido apresentados, com propostas de adaptação de linguagens existentes, e até linguagens completamente novas. Com tantas maneiras de desenvolver aplicativos multiplataforma, há a necessidade de avaliar os muitos *frameworks* existentes e realizar comparação deles com o desenvolvimento nativo. Muitos estudos têm focado somente em desenvolvimento *mobile*, havendo, no entanto, mais ambientes a serem considerados e avaliados como equipamentos ligados ao conceito *IoT*(*Internet of Things*) e até computadores tradicionais. Deste modo a proposta aqui apresentada tem como objetivo a avaliação do desenvolvimento multiplataforma, usando a linguagem *Flutter* frente o desenvolvimento nativo, se utilizando de um *framework* de avaliação que considere pesquisas acadêmicas em frente a percepção dos desenvolvedores, e ainda sim, atendendo as tendências de mercado. Após aplicados os instrumentos de pesquisa, foi verificado que por mais que ainda possua pontos negativos em relação às linguagens nativas, *Flutter* apresenta uma base estável sobre a qual é possível realizar o desenvolvimento de aplicações multiplataforma *mobile*. Por mais que seja um ambiente estável em aplicações *mobile*, como pesquisado neste trabalho, esse *framework* em outros ambientes como sistema Linux ainda está em teste e não são considerados estáveis. Assim, *Flutter* ainda não pode ser considerado um ambiente estável para desenvolvimento multiplataforma fora de ambientes *mobile*.

Palavras-chave: Desenvolvimento. Multiplataforma. *Flutter*. Comparação. Sistemas Operacionais.

ABSTRACT

With the growth in the number of operating systems, as well as the increasing variation of their versions, it is possible to notice an increase in the difficulty of maintaining existing programs, as well as in the creation of new programs. This difficulty arises both for different operating systems and for different versions of an operating system. Considering this difficulty, many ways have been presented to carry out multiplatform development. Diverse *framework*s have been presented, with proposals for adapting existing languages, and even completely new languages. With so many ways to develop multiplatform applications, there is a need to evaluate the many existing *framework*s and compare them with native development. Many studies have focused only on mobile development, there are, however, more environments to be considered and evaluated as equipment connected to the IoT concept (Internet of Things) and even traditional computers. In this way, the proposal presented here aims to evaluate multiplatform development, using the *Flutter* language against native development, using an evaluation *framework* that considers academic research in compared to the developers' perception, and yet, meeting market trends. After applying the research instruments, it was found that even though *Flutter* still has negative points in relation to native languages, its presents a stable base on which it is possible to carry out the development of multiplatform mobile applications. As much as it is a stable environment in mobile applications, as show in this work, other environments such as Linux are still being tested and are not considered stable. So, *Flutter* cannot yet be considered a stable environment for multiplatform development outside of mobile environments.

Keywords: Development. Cross-platform. *Flutter*. Comparision. Operating System.

LISTA DE GRÁFICOS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

DSRM	*Design Science Research Methodology*
FPS	*Frames per second*
GPU	*Graphics Processing Unit*
HAL	*Hardware Abstraction Layer*
IDE	*Integrated Development Environment*
IOT	*Internet of Things*
LTS	*Long Term Support*
OS	*Operating System*
RAM	*Random Access Memory*
SDK	*Software Development Kit*
TI	Tecnologia da Informação

SUMÁRIO

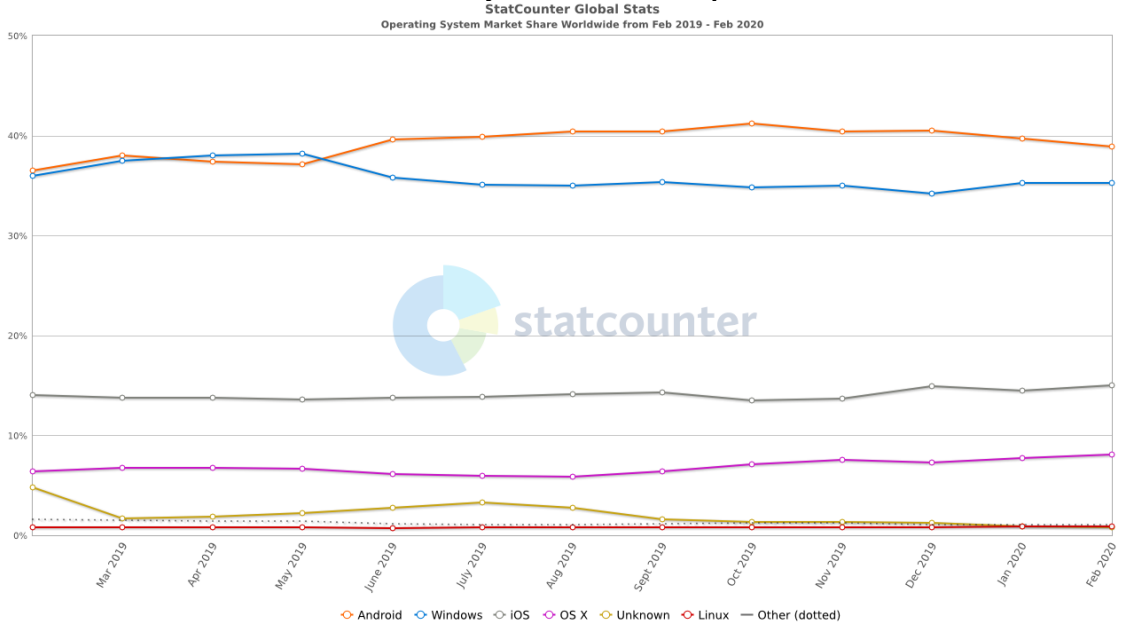
1 INTRODUÇÃO

Com a democratização da tecnologia, percebe-se um aumento nos ambientes em que um mesmo programa pode ser utilizado (LÖFFLER; GÜLDALI; GEISEN, 2010), exemplo disso são as redes sociais, que podem ser utilizadas em diferentes ambientes como celulares e computadores. Os desenvolvedores têm buscado essa disponibilidade de manutenção de código simplificada entre muitos ambientes (RIEGER; MAJCHRZAK, 2019).

No entanto pode-se verificar uma grande diferença nos 5 maiores sistemas operacionais, que são os principais fatores nas diferenças de ambiente, e isso somente considerando ambientes nativos *mobile* e de computadores (BIØRN-HANSEN *et al.*, 2019), (BOUSHEHRINEJADMORADI *et al.*, 2016) e dentro destes tem-se ainda diferenças grandes de versão (RIEGER; MAJCHRZAK, 2019).

Como é possível verificar no gráfico abaixo, tem-se como principais OS utilizados na atualidade utilizado STATCOUNTER (2019), exemplo Android 38.9%, Windows 35.29%, iOS 14.97%, OS X 8.07%, Linux 0.89%, Desconhecido 0.83%.

Gráfico 1 - **Status da distribuição de sistemas operacionais 2019**



Fonte: STATCOUNTER (2019)

Nota-se que esses ambientes têm mantido uma distribuição equivalente no último ano, e que ainda existem diferenças de versão que muito afetam a maioria dos ambientes apresentados.

Esse meio distribuído de ambientes tende a aumentar, recentemente a Huawei anunciou seu ambiente multiplataforma o *HarmonyOS*, ambiente este de sistema operacional para diferentes dispositivos(BIØRN-HANSEN; GRØNLI; GHINEA, 2019c), sendo possível o mesmo ambiente ser encontrado em um celular, carro, *wearable* e muitos outros equipamentos.

Da mesma maneira o Google busca com o *Fuchsia*, seu sistema operacional multiplataforma , tendo como objetivo apresentado a sua compatibilidade, de maneira que o ambiente possa ser compartilhado, do mesmo modo que tanto uma geladeira quanto um carro possam utilizá-lo, já que precisam de sistemas internos, que além de controle, possam ter uma interface de utilização (BIØRN-HANSEN *et al.*, 2019), (BOUSHEHRINEJADMORADI *et al.*, 2016).

Como os equipamentos basicamente estão se diversificando e há cada vez mais ligação entre eles tende-se a necessidade de trabalhar na criação de aplicativos que funcionem de maneira clara e em todos eles (RIEGER; MAJCHRZAK, 2019). Esse processo de programar em vários ambientes traz consigo algumas dificuldades.

Alguns autores destacam como dificuldade o custo necessário para o desenvolvimento de uma aplicação *mobile* nativa, já que os custos para desenvolver somente para um ambiente já podem ser altos(LÖFFLER; GÜLDALI; GEISEN, 2010), (BIØRN-HANSEN *et al.*, 2019). Quando o aplicativo deve atender a vários ambientes é necessário replicar o tempo para o desenvolvimento de cada aplicativo, por plataforma, podendo ainda o valor para cada ambiente ser diferenciado (BIØRN-HANSEN *et al.*, 2019),(BOUSHEHRINEJADMORADI *et al.*, 2016), o que gera um crescimento exponencial no custo de desenvolvimento.

Outro aspecto apresentado como dificuldade no desenvolvimento são os diferentes focos das aplicações. Muitas vezes os objetivos a serem alcançados por cada grupo de desenvolvimento podem ser completamente diferentes, analisando por exemplo um aplicativo *mobile*, que normalmente foca em experiência de usuário e em outras situações em rentabilidade, enquanto um ambiente voltado para carros, precisaria mais foco em segurança (RIEGER; MAJCHRZAK, 2019).

Do mesmo modo há outras características que precisam ser consideradas, como a diferença entre os ambientes de desenvolvimento, a metodologia de testes, a

avaliação de usuários, o suporte e validação, entre outros, que dificultam a realização de um projeto estável e contínuo em um ambiente de tecnologia.

Múltiplos ambientes como demonstrado acima adicionam complexidade a um projeto de desenvolvimento, no entanto, pelo ponto de vista dos usuários a facilidade de uso e fluidez, que são essenciais para uma aplicação (LÖFFLER; GÜLDALI; GEISEN, 2010), para uma escolha baseada em fatos é necessário analisar as ferramentas que são feitas para ambientes multiplataforma, focados em reutilização de código, se possível completa.

Uma das linguagens de programação que vem se mostrando uma possibilidade de resposta a esse cenário é o *Flutter*. *Flutter* é um *framework* multiplataforma, cujo foco inicial seria entregar a capacidade de que, com o mínimo de alteração de código e com a maior velocidade e fluidez na utilização final da aplicação, ainda seja possível desenvolver uma mesma aplicação para vários ambientes.

Sendo apresentada pela primeira vez em 2015, e com sua primeira versão estável em 2019 (BIØRN-HANSEN; GRØNLI; GHINEA, 2019b), (JIM, SIMON *et al.*, 2020). O *Flutter* utiliza a linguagem *DART* que apresenta semelhanças a desenvolvimento de linguagens conhecidas, como C e Java, e ainda apresenta e documentação ampla. Este *framework* tem se apresentado como o mais fluido e preparado modelo de desenvolvimento multiplataforma (BIØRN-HANSEN *et al.*, 2019), (BOUSHEHRINEJADMORADI *et al.*, 2016).

Essa capacidade tem sido demonstrada por muitas linguagens no meio atual (RIEGER; MAJCHRZAK, 2019), e por mais que muitas estejam sendo utilizadas, o foco tem sido em ambientes *mobile*, assim como o foco das análises feitas até o momento. Vendo esta situação se faz necessário avaliar linguagens em que o modelo de programação e o suporte a ambientes sejam multiplataforma por completo(LÖFFLER; GÜLDALI; GEISEN, 2010).

Para esse meio de análise precisa-se reconhecer que não somente a necessidade do desenvolvedor precisa ser levada em conta, que já considera utilização de recursos nativos e escalabilidade(BIØRN-HANSEN *et al.*, 2019), mas também segurança, estabilidade e performance (BIØRN-HANSEN *et al.*, 2019),(BOUSHEHRINEJADMORADI *et al.*, 2016), (RIEGER; MAJCHRZAK, 2019).

Para que uma análise completa e efetiva seja realizada, é necessário utilizar um modelo de análise para desenvolvimento multiplataforma conforme apresentado por Christoph Rieger e Tim A. Majchrzak (2019) no qual deve considerar:

- Necessidades de um desenvolvedor para diminuição de codificação, ainda sim considerando a facilidade e familiaridade do desenvolvedor com a linguagem.

- Capacidade de utilizar, da maneira mais simples possível, funções nativas da plataforma.

- Considerar outras plataformas e não somente *mobile* no desenvolvimento.

- Experiência do usuário na utilização do app desenvolvido pela plataforma.

Hoje por padrão, modelos utilizam somente as plataformas *mobile* como base para a análise.

Para tal análise tende-se a necessidade de levar em conta múltiplos fatores que acarretam pesos diferentes para cada necessidade, podendo ser analisado ambientes de multiplataforma entre si, e com a programação nativa (BOUSHEHRINEJADMORADI *et al.*, 2016), (BIØRN-HANSEN; GRØNLI; GHINEA, 2019a), (BIØRN-HANSEN *et al.*, 2019).

Pode-se assim verificar que é possível utilizar *frameworks* de desenvolvimento multiplataforma para solucionar alguns dos problemas atuais, porém cada solução necessita de componentes e comportamentos específicos, o que leva a necessidade de avaliar qual linguagem ou *framework* é mais adequado.

A escolha necessita ser alicerçada em análises sólidas e concisas. Sendo assim este estudo traz como proposta analisar o *Flutter* como *framework* que desenvolva com uma única linguagem para Android, IOs, Windows e Linux, e compará-lo ao desenvolvimento nativo em relação a usabilidade, capacidade de uso de funções nativas e velocidade.

1.1 Metodologia

Considerando a construção do conhecimento, é necessário seguir princípios, práticas e procedimentos que são comumente aceitos, a metodologia conhecida como *Design Science Research Methodology* (DSRM) apresenta passos que se seguidos podem criar artefatos que realizem objetivos (PEFFERS *et al.*, 2007), (JUNIOR *et al.*, 2017).

Para aplicação desse modelo, foram seguidas seis etapas:

- Etapa 1: Identificação do problema e sua motivação, analisando o problema apresentado, seu histórico e demonstrando qual a pergunta este trabalho responderá.

- Etapa 2: Definição dos objetivos para a solução, nessa etapa com o conhecimento do problema em mãos, bem como o fato de ser um problema real e sua solução agregar conhecimento necessário, colocam-se os objetivos da solução, assim como o estado da arte de soluções previamente apresentadas.
- Etapa 3: Projetar e Desenvolver, nesta etapa os artefatos serão desenvolvidos, sua funcionalidade será determinada assim como sua arquitetura. Isso também envolve o conhecimento da teoria que será utilizada para solução.
- Etapa 4: Demonstração, neste passo serão apresentados os usos dos artefatos desenvolvidos, por meio de experimento e simulação.
- Etapa 5: Avaliação, nesta etapa os resultados serão mensurados e, para este trabalho, alguns métodos de avaliação específicos serão utilizados. A avaliação conterá componentes Analíticos de Otimização e Análise Dinâmica, além do experimento conter simulações, experimentos controlados e cenários para demonstrar sua utilidade.
- Etapa 6: Comunicação, neste momento serão apresentados os problemas e a proposta solução, além dos artefatos desenvolvidos.

É importante frisar que durante este trabalho não será desenvolvido uma aplicação, somente a prototipação de uma, com objetivo de demonstrar pontos específicos da interface de uma aplicação. Os artefatos terão como foco os trabalhos apresentados em capítulos posteriores.

1.2 Objetivos

Como demonstrado na Introdução, foi apresentado o problema em relação a desenvolvimento em muitos ambientes, sendo assim o objetivo geral deste trabalho é de analisar o *framework Flutter*, e sua aplicação para desenvolvimento em múltiplas plataformas com seus diversos sistemas operacionais, frente ao desenvolvimento nativo.

Assim se espera durante o trabalho que se apresente um *framework* de avaliação multiplataforma, seja desenvolvido um modelo de análise específico para

avaliação de *Flutter*. Primeiramente no capítulo dois será apresentado a revisão sistemática na qual é baseado a metodologia utilizada para avaliação. No capítulo três é apresentada os principais ambientes apresentados e no quatro as bases para utilização da metodologia. No capítulo cinco a metodologia é apresentada detalhadamente e o capítulo seis é apresentado os resultados de sua aplicação, chegando então à conclusão do trabalho.

2 TRABALHOS RELACIONADOS

Para a busca de trabalhos relacionados optou-se por uma revisão sistemática, este capítulo descreve uma revisão sistemática com o objetivo de apoiar o presente estudo, buscando artigos que agregassem um modelo de avaliação de linguagens multiplataforma. Conforme apresentado por (KITCHENHAM *et al.*, 2009) a revisão deve ser norteada pela questão de pesquisa, sendo que neste caso, esta questão é apresentada como: Busca por avaliação de linguagens multiplataforma.

2.1 Protocolo

O protocolo de avaliação define os procedimentos para a avaliação sistemática da literatura, formalizando um registro para realização da revisão. O modelo utilizado está baseado em (KITCHENHAM, B.; KITCHENHAM; CHARTERS, 2007), tendo como foco a área da computação. Na sequência são apresentados os detalhes dos protocolos seguidos.

Esta revisão sistemática teve como objetivo pesquisar e avaliar os modelos de avaliação de *framework*s multiplataforma, servindo de subsídio balizador para o desenvolvimento deste trabalho de conclusão. É importante destacar que o termo multiplataforma, utilizado nesta pesquisa não está preso apenas a ambientes *mobile*.

Para o auxílio na criação da *string* de busca (KITCHENHAM, B.; KITCHENHAM; CHARTERS, 2007) recomendam a utilização do PICOC (População, Intervenção, Comparação, Resultados e Contexto). A formulação da pesquisa é apresentada a seguir.

2.1.1 Formulação da Pesquisa

a) Foco da questão

Esta revisão sistemática busca artigos que criem avaliações para serem utilizadas entre linguagens multiplataforma, e entre nativo e multiplataforma.

b) Questões de interesse:

1. Dispositivos utilizados para avaliação;

2. Dificuldades dos desenvolvedores;
3. Dificuldades do usuário;
4. Estudos já realizados;

c) Palavras-chaves:

Cross-platform, Flluter, comparisons, native development.

d) Intervenção:

Verificar as tecnologias que são utilizadas nas pesquisas.

e) Controle:

Não será utilizado.

f) Efeito:

Identificar um *framework* para avaliação de desenvolvimento multiplataforma.

g) Medida do Resultado:

Gerar um embasamento teórico para a dissertação, assim como a publicação de artigos científicos.

h) População de interesse:

Pesquisadores, professores e desenvolvedores.

i) Aplicação:

Esta revisão sistemática tem como foco pesquisadores, professores e alunos da área da computação e desenvolvimento voltado a multiplataforma.

j) Desenho do experimento:

Não será desenvolvido.

k) Financiamento:

Não há.

2.1.2 Base de Dados

a) Definição dos critérios de seleção das bases:

As fontes de dados foram selecionadas através da indicação dos orientadores deste projeto e de serem as bases que armazenam trabalhos referenciados pela comunidade acadêmica. A base de pesquisa *Google Scholar* oferece acesso a pesquisas confiáveis, íntegras e multidisciplinares, tendo conteúdo integral apenas para pesquisadores através do portal da CAPES (Web Of Science, 2018); já o *IEEE Xplore* fornece acesso a algumas das publicações mais citadas no mundo em Ciência da Computação, Eletrônica e Engenharia Elétrica (IEEEXPLORE, 2018).

b) Idiomas das Fontes de Dados:

Somente serão consideradas as publicações nas línguas inglês e português.

c) *String* de busca:

A *string* de busca utilizada foi criada com os seguintes elementos obrigatórios da pesquisa: "*cross-plataform*" e "*Flutter*" ela foi associada aos termos *comparision*, *native development*, que são termos ligados a linguagem de desenvolvimento a ser avaliada e a avaliação de *framework*s multiplataforma em si. A *string* será aplicada nas bases de dados definidas anteriormente através da sintaxe: *((Cross-plataform) or (Flluter)) and (comparision) and (native development).*

d) Artigos de controle:

Optou-se por não utilizar nenhum artigo de controle para esta revisão sistemática.

2.1.3 Seleção dos Estudos

a) Definição de estudos

i. Critérios para a inclusão/exclusão dos resultados

As publicações selecionadas devem enquadrar-se nos seguintes critérios:

a) Ano de publicação do artigo deve estar dentro do período de 2015 e 2020, buscando o estado da arte;

b) Ser um artigo científico publicado;

c) Estar escrito em inglês ou português;

d) Apresentar uma forma de validação;

e) A publicação deve estar disponível na íntegra na internet ou disponível através de convênios das instituições de ensino;

f) Ter três trabalhos fontes ou mais.

ii. Definição dos tipos de estudo

Foram selecionados estudos dos tipos teóricos, qualitativos e quantitativos referentes aos temas.

iii. Procedimentos para seleção dos estudos

Inicialmente foi utilizada a *string* de busca nas bases de dados apresentadas anteriormente. Todos os resultados, em todas as bases de dados, foram baixados e avaliados um a um.

iv. Fases de seleção de artigos:

Fase 1 - Validar os critérios de inclusão e exclusão;

Fase 2 - Leitura do título, palavras-chave e resumo;

Fase 3 - Leitura da introdução e conclusão;

Fase 4 - Leitura integral dos artigos e validação das respostas para as perguntas.

v. Critérios de qualidade das fases da Revisão Sistemática:

Os critérios de qualidade que foram avaliados nesta revisão sistemática estão descritos a seguir:

1. Que tipos de metodologias foram usadas?
2. Quais autores foram a base para a fundamentação teórica?
3. Há explicação de todos os pontos escolhidos para avaliação?
4. A avaliação pode ser usada em todo ambiente multiplataforma?
5. É proposto como realizar a avaliação?

vi. Análises adicionais:

Não há.

2.2 Desenvolvimento da Revisão Sistemática

Durante a pesquisa no Google Scholar houve 250 artigos relacionados com a *string* escolhida, primeiramente foram excluídos artigos com data anterior a 2015, diminuindo a pesquisa para 179 artigos, destes aqueles que focam em comparação entre os itens somente 125 foram encontrados. Destes aqueles que focam em comparação aplicável a linguagem *Flutter*, encontram-se 36 artigos.

Dos artigos restantes, 26 foram desconsiderados por tratarem de defesas de teses, e não artigos científicos, um dos critérios de exclusão. Dos restantes 3 foram eliminados por não estarem em inglês ou português.

A partir dos 6 restantes iniciou-se uma análise avaliativa em relação a abstract apresentado, primeiramente o trabalho de (MASCETTI *et al.*, 2020) foi desconsiderado, pois trazia comparações que aplicam-se somente a um modelo de avaliação de linguagens especificas, não podendo ser adaptado a outros modelos.

Dos restantes (SHAH; SINHA; MISHRA, 2019), (NUNKESSER, 2018) foram excluídos pois a publicação não estava disponível na íntegra na internet ou disponível através de convênios das instituições de ensino, esse processo ocorreu como demonstrado na tabela a seguir.

Tabela 1 - Filtros realizados na revisão sistemática

Passos	Ação	Quantidade de Artigos Excluídos
1	Artigos com data anterior a 2015.	71
2	Foco em Comparação.	54
3	Aplicável a linguagem *Flutter*.	89
4	Monografias, não artigos.	26
5	Não estar em inglês ou português	3
6	Não estar disponíveis gratuitamente	2
7	Leitura do abstract	2

Fonte: DO AUTOR (2020)

2.2.1 Trabalhos Para Leitura Completa

No estudo apresentado por ROMAN (2020), inicia sua análise em relação aos desenvolvimentos trazendo explicação sobre as linguagens que irá abordar, traz também seu resultado considerando uma análise qualitativa. Em relação as perguntas feitas pelo critério de qualidade, as metodologias não ficaram claras.

As bases para a fundamentação teórica são poucas e baseadas em artigos mais antigos. Os pontos escolhidos não têm artigos de comprovação e não é proposto como realizar a avaliação que ele relatou. Com as razões anteriores apresentadas este trabalho foi desconsiderado para essa análise.

O trabalho apresentado por Biørn-Hansen; Grønli; Ghinea(2019b) apresenta uma avaliação quantitativa de múltiplas linguagens, trazendo análises realizadas a partir de usos de recurso de software e hardware.

- Para avaliar as linguagens foi utilizado um modelo com três perguntas: As métricas de desempenho incluídas cumprem seu proposito em pesquisar interfaces animadas pelos usuários em aplicações moveis?
- Quão bem as ferramentas oficiais de avaliação de desempenho atendem a análise de perfil da animação e transição de desempenho nos aplicativos multiplataforma desenvolvidos?
- Qual das plataformas, *iOS* ou *Android*, exigia a menor quantidade de recursos de hardware dos dispositivos para executar animações e transições de alto desempenho?

Com essas perguntas seria possível seguir o modelo de avaliação apresentado pelos autores. Todos os pontos escolhidos estão apoiados por trabalhos de pesquisa publicados.

Com esse modelo são avaliados FPS, uso de CPU, uso de memória (RAM) e uso de GPU. Para realizar essas avalições é criado um programa em cada uma das linguagens a ser avaliada, criando uma animação e avaliando por meio de ferramentas desenvolvidas e disponibilizadas na plataforma *GITHUB* (BIØRN-HANSEN; GRØNLI; GHINEA, 2019c).

Tendo esses questionamentos como base é possível seguir o modelo de avaliação apresentado pelos autores no qual todos os pontos escolhidos estão apoiados por trabalhos de pesquisa publicados.

Este trabalho, no entanto, busca avaliar somente linguagens multiplataforma no âmbito de plataformas *mobile*, não se estendendo em outros ambientes. Como o próprio método apresentado informa as ferramentas escolhidas focam em um ambiente *mobile*, analisando de maneira quantitativa a programação. Essas análises podem ser replicadas, no entanto, para outros ambientes, sendo possível assim repetir os testes buscando diferentes ferramentas para a análise em outros ambientes como desktop ou web.

Já no trabalho apresentado por Rieger(2019), é possível verificar que os autores buscam criar um *framework* cujo o objetivo é avaliar linguagens multiplataforma, não considerando somente o ambiente *mobile* mas também outros ambientes como *weareable* e outros.

Nesta análise a metodologia criada está baseada em outras pesquisas, com cada ponto de avaliação escolhido baseado em um ou mais trabalhos, trazendo embasamento teórico para o *framework*. É possível aplicar o *framework* a situações especificas, pois o artigo apresenta uma maneira de utilizá-lo assim como cria um sistema de métricas que pode ser adaptado a necessidade do avaliador.

2.3 Trabalhos Correlatos não ligados a Revisão Sistemática

Assim como esse trabalho, outros trabalhos procuraram avaliar linguagens multiplataformas, como pode ser visto no trabalho apresentado por (WU, 2018) tenta

demonstrar a comparação entre *REACT NATIVE* e *Flutter* para isso ele apresenta uma pesquisa sobre conceitos fundamentais sobre a plataforma *Flutter* e *REACT NATIVE*, demonstrando característica como o uso de JSX no código do *REACT NATIVE* e como essa linguagem interage com a plataforma por meio do DOM. Além disso traz noções do funcionamento da linguagem e utilização em programação.

Após esta etapa o autor apresenta a linguagem que será usada na comparação, *Flutter*, e suas principais características como uso de *widget* e controle de estado.

Tendo como base estes princípios, é selecionado um caso de estudo, e a partir desse caso de estudo é desenvolvida nas duas linguagens, e posteriormente comparados, com relação a *layout,* estilo e performance

Em especial, no quesito performance, é analisada a quantidade de FPS durante a rolagem de página, pois ao iniciar uma rolagem a página deve apresentar resposta instantânea, ao menos a percepção do usuário, e medir a quantidade de FPS nesse momento apresenta um bom modelo de avaliação para percepção do usuário.

Além do tempo de resposta, também foi analisado o tempo de leitura e escrita em arquivos da aplicação, o tempo de escrita foi avaliado utilizando um programa específico, em que se abria um arquivo e se grava e se lê informação, cronometrando cada passo, com várias tentativas documentadas para comparação.

Pode-se ver assim como (WU, 2018) em seu trabalho se propôs a comparar as plataformas *Flutter* e *REACT NATIVE,* buscando uma análise cujo embasamento pode ser utilizado para comparar linguagens de forma quantitativa, em relação a uso de hardware.

Para esse trabalho, no entanto, a comparação que foi feita em relação a desenvolvimento nativo e utilização do conceito de maneira mais ampla, já que trabalhos anteriores mantiveram o foco em desenvolvimento multiplataforma somente entre plataformas *mobile*.

3 CARACTERISTICAS DAS PLATAFORMAS DE DESENVOLVIMENTO

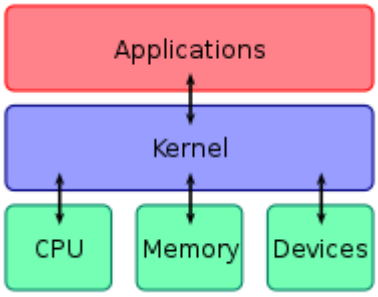
O desenvolvimento multiplataforma não é uma ideia nova, (ARNOLD; GOSLING; HOLMES, 2005) apresentam no conceito de máquina virtual o funcionamento da linguagem JAVA, esta máquina virtual, com os pré-requisitos instalados, faria com que um programa pudesse funcionar tanto em ambientes Windows como em ambientes Linux ou Mac OS, tornando assim JAVA uma linguagem multiplataforma.

Atualmente com o mercado de celulares sendo superior ao de outras plataformas, o desenvolvimento multiplataforma tem seu foco em ambientes *mobile*, o que aumenta a quantidade de *framework*s de desenvolvimento multiplataforma para esse ambiente. Dos ambientes *mobile* os que se destacam são os ambientes *Android* e *IOS* (STATCOUNTER, 2019).

3.1 Ambiente Android

O *Android* foi lançado em novembro de 2007, tendo como objetivo ser uma plataforma *open-source* para desenvolvimento *mobile* (OZGUL *et al.*, 2010), (GRONLI *et al.*, 2014). O *Android* é uma pilha de software com base em kernel *Linux open-source* que foi criada para uma gama de dispositivos(GOOGLE, 2020). A base do *Android* é um kernel Linux, que tem como objetivo integrar o sistema operacional com os dispositivos de hardware, assim como facilitar sua interações (THE LINUX INFORMATION PROJECT, 2004) como é possível verificar na Figura 1.

Figura 1 - **Funcionamento do Kernel**



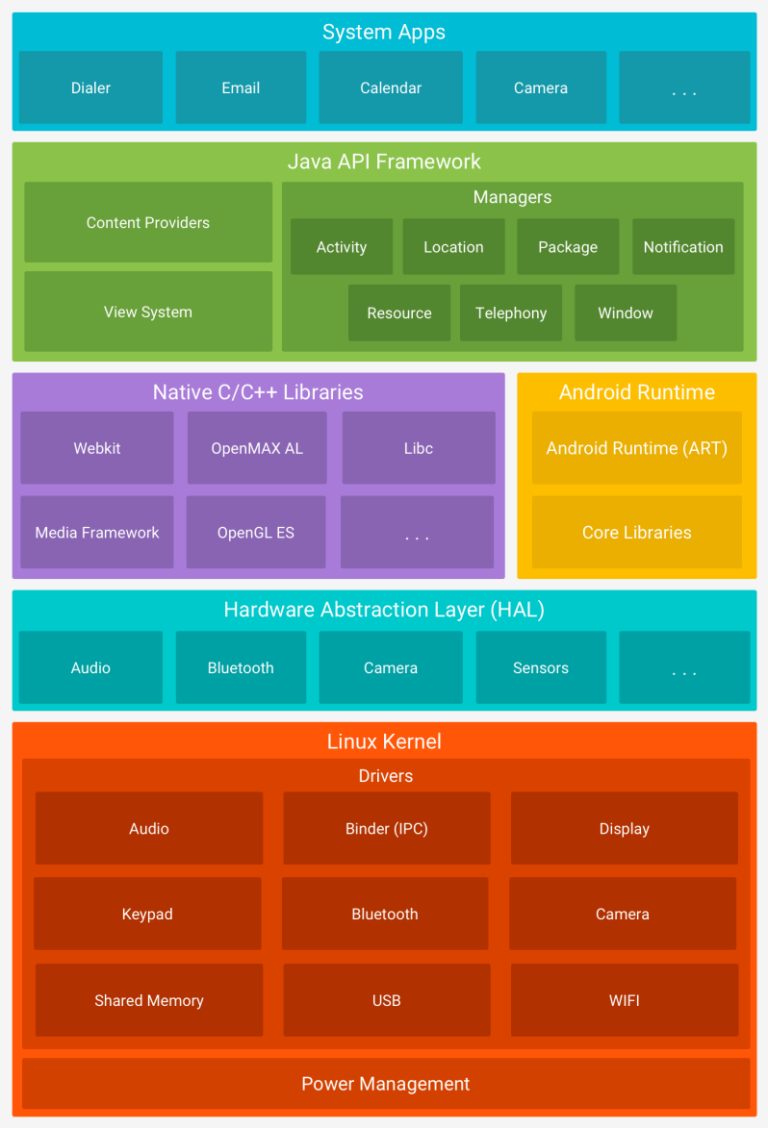
Fonte: KERSHELL (2011)

A camada de abstração de Hardware do *Android* foi lançada em novembro de 2007, tendo como seu objetivo o de ser uma interface padrão para o melhorar a exposição dos dispositivos de hardware com o nível mais alto do sistema(GOOGLE, 2020).

Após essa camada existem as camadas do *Android Runtime,* que cria uma máquina virtual parar que cada aplicação rode em um ambiente controlado próprio, e a camada de bibliotecas nativas C e C++, que podem ser acessadas diretamente ou por meio da camada superior, e contém funcionalidades padrão para o ambiente.

A última camada antes das aplicações e a camada da JAVA API, que é a base da programação nativa *Android*, dando todos os recursos para a programação nativa. A Figura 2 exemplifica este ambiente:

Figura 2 - **Diagrama dos componentes do Android**



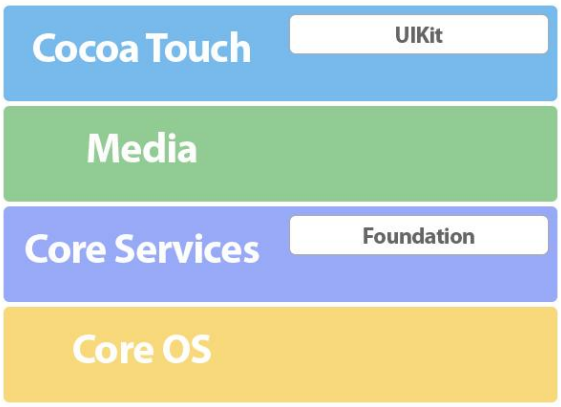
Fonte: GOOGLE (2020)

3.2 Ambiente *iOS*

O sistema operacional *iOS* é o ambiente *mobile* para os dispositivos da *Apple*, assim como para outros dispositivos dessa empresa. Aplicações em *iOS* são escritas em *OBJECTIVE-C* que é uma extensão da linguagem C, que suporta programação orientada a objetos(GRONLI *et al.*, 2014).

A camada do *Core OS* contém as funcionalidades básicas do sistema, assim como encapsula o *kernel*, enquanto a camada de *Core Services* provê o gerenciamento de serviços que as aplicações podem utilizar. A camada de *Media* contém as funções de áudio e mídia para as aplicações e a camada de *Cocoa Touch* contém a base para uma aplicação *iOS*. O diagrama pode ser verificado na imagem abaixo (Figura 3):

Figura 3 - **Diagrama dos componentes do iOS**



Fonte: SR. IOS DEVELOPER (2018)

No entanto com o crescimento do mercado de *IoT*, por mais que o mercado *mobile* ainda se mantenha como principal mercado, se verifica aumento na participação de lucros para *IoT*, principalmente nos ramos industriais com sensores (EYGM LIMITED, 2019). A partir disso as linguagens multiplataforma também devem se preparar para um ambiente *IoT*.

3.2 Desenvolvimento Multiplataforma com *Flutter*

O *Flutter* iniciou como um experimento por desenvolvedores do Google que tentavam remover camadas de suporte de compatibilidade do navegador Chrome,

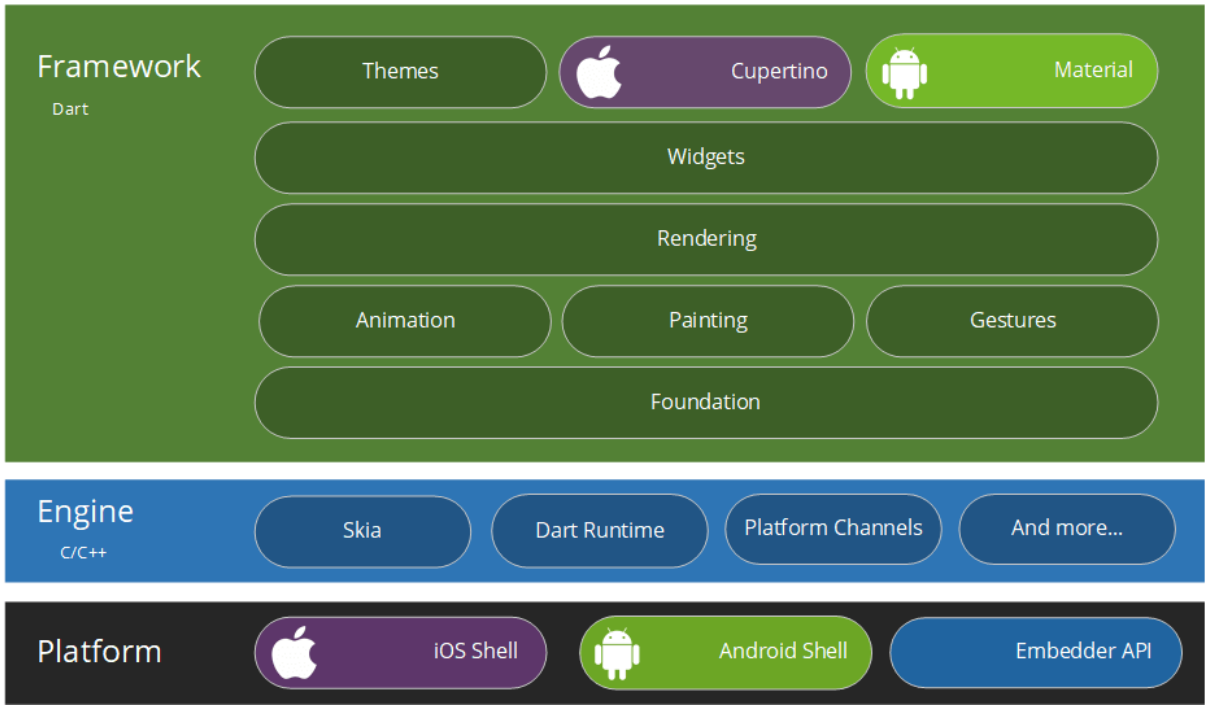
tentando fazer com que ele rodasse mais rapidamente. Após algumas semanas de teste verificaram que o resultado de seus testes processava vinte vezes mais rápido que o Chrome(MAINKAR; GIORDANO, 2019).

Flutter é um *framework* para desenvolvimento multiplataforma, suporta atualmente desenvolvimento *mobile* e *PC.* Também está em seu *roadmap* suportar desenvolvimento *web* (MAINKAR; GIORDANO, 2019), (JIM, SIMON *et al.*, 2020).

O desenvolvimento *Flutter* utiliza a linguagem *DART. DART* é uma linguagem que segue o modelo da linguagem C, é orientada a objeto, baseada em classes, com o sistema de tipo opcional e com heranças(GOOGLE, 2013), (BRACHA, 2015).

Ao contrário de outras linguagens multiplaforma, o *Flutter* não tem uma lista de componentes que tem equivalentes na linguagem nativa, mas tem sua própria *engine* de renderização. Essa *engine* e compilada através da *Android NDK,* e o código *DART* é compilado em código nativo, esse processo utiliza o *SKIA* para realizar o desenho da interface e compila os *Widgets* em nativo(MAINKAR; GIORDANO, 2019), (JIM, SIMON *et al.*, 2020). É possível ver na Figura 4 o diagrama da linguagem:

Figura 4 Figura 4 - Diagrama dos componentes do *Flutter*

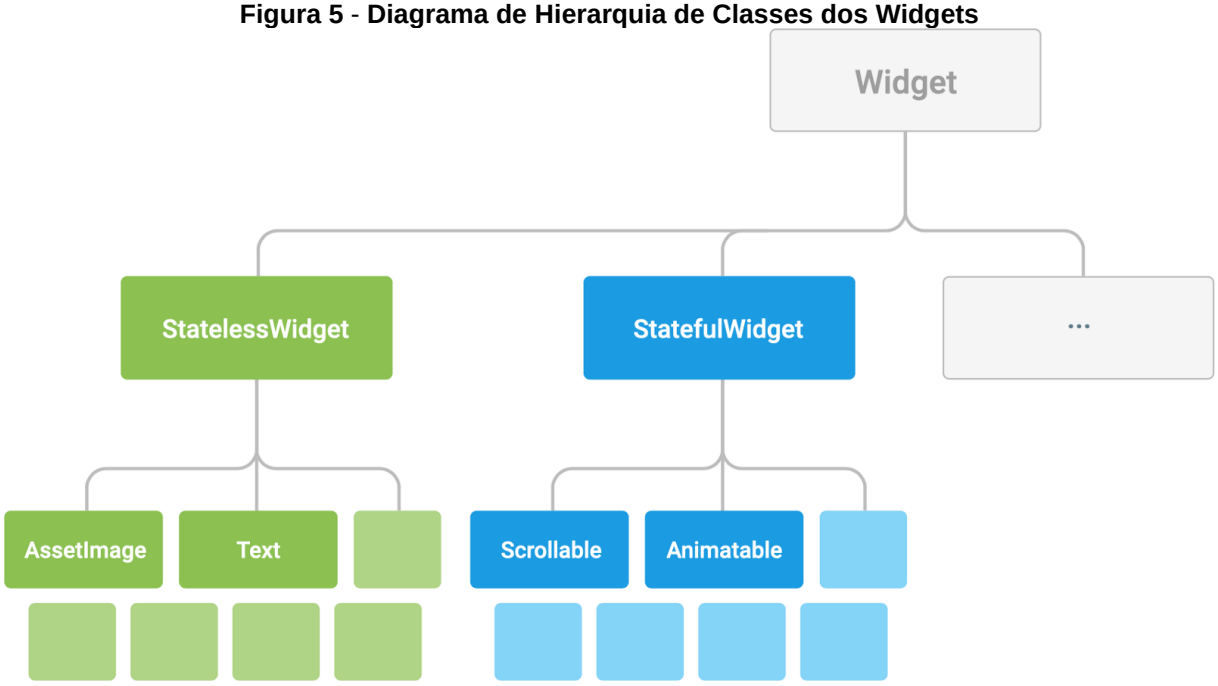


Fonte: JIM, SIMON (2020)

3.2.1 Widgets

Em *Flutter* tudo é um *widget*, desde botões, imagens, listas, tudo é feito como *widget*. Os *widgets* substituem todos os componentes nativos, e estarão presentes por toda a aplicação. Em suma o *widget* pode definir um elemento estrutural como um botão, um elemento de estilo como uma fonte, um aspecto de layout como uma borda, e assim por diante. Ao criar um *widget* pensando no layout, é importante verificar que existem dois tipos de *widgets*, os *Stateless*, que não te estado e os *Statefull,* cujo estado pode ser atualizado (SHAH; SINHA; MISHRA, 2019), (TRAN, 2020), (JIM, SIMON *et al.*, 2020) como demonstrado na Figura 5.

Figura 5 - **Diagrama de Hierarquia de Classes dos Widgets**



Fonte: JIM, SIMON (2020)

No entanto para avaliar de maneira mais assertiva a linguagem *Flutter* é necessário buscar um modelo no qual se basear.

4 PRINCIPAIS ASPECTOS PARA AVALIAÇÃO

Considerando as possibilidades em desenvolvimento multiplataforma apresentadas em capítulo anterior, propõe-se avaliar em relação ao nativo a linguagem *Flutter*. Para uma satisfatória avaliação foi realizada uma revisão sistemática, a partir desta revisão foi possível encontrar um artigo que, baseado nos pontos de aceitação da revisão, pode ser utilizado para avaliação de linguagens multiplataforma.

Neste modelo alguns fatores precisam ser considerados, esses fatores serão utilizados de base para a avaliação. Este modelo precisa conter o ponto de vista de usuários, avaliadores, empresas e qualquer significativo padrão que possa alterar a escolha de que linguagem utilizar (RIEGER; MAJCHRZAK, 2019).

4.1 Avaliação do Ponto de Vista do Usuário

Como apresentado por (LÖFFLER; GÜLDALI; GEISEN, 2010) , é importante perceber que a percepção do usuário se tornou muito mais que somente um relato a ser adicionado ao final dos desenvolvimentos, as avaliações postadas em diversas *app stores* permeiam os algoritmos que alavancam o aplicativo nas ordenações de pesquisas (BIØRN-HANSEN *et al.*, 2019),podendo acarretar em maior visibilidades para aplicações pagas.

Mesmo aplicações gratuitas recebem esse modelo de avaliação de *stores* como a *App Store*, *Play Store* (BIØRN-HANSEN; GRØNLI; GHINEA, 2019a), (BOUSHEHRINEJADMORADI *et al.*, 2016) e *Microsoft Store*. Assim a avaliação de um usuário deve ser analisada ao considerar a escolha entre desenvolvimento nativo e multiplataforma.

4.1.1 Tempo de resposta

Os parâmetros em que o usuário sentiria a diferença entre os desenvolvimentos seriam dados a observações não técnicas (RIEGER; MAJCHRZAK, 2019). Entre essas observações pode-se considerar o tempo de resposta da interface de usuário, (LÖFFLER; GÜLDALI; GEISEN, 2010) o que normalmente está ligado à performance da aplicação no ambiente onde ela está instalada.

4.1.2 Confiabilidade

Outro ponto de avaliação necessário a ser verificado em relação ao usuário é a confiabilidade da aplicação durante uso (BIØRN-HANSEN *et al.*, 2019), (RIEGER; MAJCHRZAK, 2019), com a quantidade de erros ocorrendo em uso afetando esse ponto. Para exemplificar essa situação é possível considerar um usuário em que a internet seja intermitente, com uma aplicação que utiliza a internet também de maneira intermitente, nessa situação a linguagem disponibiliza tratamentos preparados para que os erros não piorem a experiência do usuário.

4.2 Avaliação do Ponto de Vista do Desenvolvedor

O desenvolvedor é um ponto central na avaliação de aplicações multiplataforma, seu desenvolvimento será diretamente afetado pelas capacidades da linguagem escolhida, assim como na participação da escolha de que linguagem utilizar para o desenvolvimento do projeto (BIØRN-HANSEN; GRØNLI; GHINEA, 2019a), (BOUSHEHRINEJADMORADI *et al.*, 2016). Em razão disso um modelo de avaliação sob o ponto de vista do desenvolvedor é um critério importante e essencial para sua escolha (RIEGER; MAJCHRZAK, 2019).

4.2.1 Ferramentas de programação

Alguns aspectos que o desenvolvedor adquire durante sua trajetória são importantes, ao utilizar uma nova linguagem, algumas facilidades aprendidas com outras linguagens como integrações com IDE’s, auto completar, atalhos entre outros precisam ser levados em conta(FORD; PARNIN, 2015). Essas diferenças aumentam a curva de aprendizado, incrementando a resistência de desenvolvedores, além de dificultar adaptação, o que acrescenta frustação ao desenvolvedor(FORD; PARNIN, 2015), (RIEGER; MAJCHRZAK, 2019).

4.2.2 Escalabilidade

Desenvolver uma aplicação em grande escala e com uma grande quantidade de desenvolvedores demanda que projetos complexos sejam separados em várias partes, e cada desenvolvimento destas partes seja feito por grupos diferentes. A linguagem precisa estar pronta para ser desenvolvida em partes. Ao responder uma pergunta em um blog, o engenheiro da Microsoft Axel Rietschin, informou que após uma semana sem enviar uma alteração por meio de um aplicativo de versionamento, sua alteração ficou atrás de 60 mil novas atualizações (QUORA, 2020).

Quanto maior a complexidade de um projeto e maior a quantidade de envolvidos a capacidade de escalabilidade de programação deve ser considerada, já que a linguagem pode afetar o modo como o trabalho é dividido e atualizado entre os desenvolvedores (RIEGER; MAJCHRZAK, 2019).

4.2.3 Testes

As validações feitas ao finalizar uma programação confirmam a possibilidade de entrega de uma aplicação. Criar diferente versões de uma mesma aplicação, de maneira que esta rode em diferentes ambientes, e mesmo assim funcione da mesma maneira adiciona complexidade aos testes dessas aplicações (BOUSHEHRINEJADMORADI *et al.*, 2016), (KLUBNIKIN, 2019).

Além da frustação da necessidade de redesenvolvimento (FORD; PARNIN, 2015),a falha em criar *scripts* para testes automatizados pode atrasar o cronograma de uma equipe ágil(LÖFFLER; GÜLDALI; GEISEN, 2010). Em razão disso a capacidade de automatizar teste, criar e realizar testes unitários se torna parte importante na avaliação de uma linguagem, assim também sendo necessário ser avaliado entre nativo e multiplataforma.

4.2.4 Acesso a funcionalidades nativas

Durante o desenvolvimento, em diversas situações pode ocorrer a necessidade de a aplicação ter acesso às funções nativas do ambiente em que ela se encontra, funções como *GPS*, giroscópio e medidor de temperatura.

Além de acesso a equipamentos que se encontram fisicamente no equipamento, também é necessário que a aplicação possa se conectar as funcionalidades de software do ambiente. Entre essas funcionalidades pode-se

encontrar acesso ao banco de dados, criação e pesquisa de arquivos, acesso a aplicações que são nativas do ambiente como o *google maps* no caso do *android*.

Em virtude destas características é possível observar que, este acesso já existe nas linguagens nativas da plataforma, o que se evidencia como um aspecto importante a ser considerado, e que precisa também se encontrar nas linguagens multiplataformas (BIØRN-HANSEN; GRØNLI; GHINEA, 2019b).

4.3 Outros Fatores de Avaliação

Além dos pontos já apresentados, é possível observar que uma linguagem multiplataforma afeta a aplicação de outras maneiras. Entre essas maneiras as consideradas para a escolha de um modelo têm-se a Segurança, Suporte, Configuração e Plataformas em que pode ser utilizada(BIØRN-HANSEN *et al.*, 2019), (BOUSHEHRINEJADMORADI *et al.*, 2016), (RIEGER; MAJCHRZAK, 2019).

4.3.1 Segurança

A segurança dos dados que uma aplicação tem acesso é um fator de importância singular. Hoje um ambiente *mobile* ou *desktop* pode conter informações que disponibilizam acesso a controle financeiro, opções de escolha pessoal, informações familiares e outros dados sensíveis dos seus utilizadores. Em razão disso a linguagem precisa garantir que seu modelo não trará riscos à segurança do ambiente onde se encontra (BIØRN-HANSEN *et al.*, 2019), (BOUSHEHRINEJADMORADI *et al.*, 2016).

Além disso ambientes podem adicionar maiores cuidados com esse requisito específico. Uma aplicação com acesso ao computador de bordo de um carro teria maior peso nesse requisito que em um ambiente *mobile,* já que uma falha de segurança nesse ambiente acarretaria um risco a vida de quem o utiliza (RIEGER; MAJCHRZAK, 2019).

4.3.2 Suporte

Quando uma nova linguagem é escolhida, poderá haver dúvidas por parte dos desenvolvedores, o que acarreta a necessidade de suporte ao desenvolvimento.

Linguagens de desenvolvimento necessitam de suporte a aplicação, linguagens ligadas a empresas podem ter suporte pago para desenvolvimento.

Em projetos de código aberto normalmente comunidades que ajudam a retirar dúvidas e partilhar experiencias no desenvolvimento utilizando a linguagem. Havendo uma empresa que preste suporte ou uma comunidade, o suporte ao desenvolvimento é um ponto que precisa ser analisado ao realizar a escolha(HUDLI; HUDLI; HUDLI, 2015).

4.3.3 Configuração e plataformas em que pode ser utilizada

Avaliações entre linguagens multiplataforma e nativa tendem a avaliar somente ambientes *mobile*, sendo assim o modelo de avaliação escolhido precisa ser capaz de avaliar outros ambientes além do *mobile*. Ambientes como *wearables* e um sistema de controle de um carro também precisam ser avaliáveis. Esses ambientes, assim como os ambientes de IoT se tornam cada vez mais comuns e presentes no mercado(RIEGER; MAJCHRZAK, 2019).

5 MODELO DE AVALIAÇÃO

Com base na revisão da literatura, foi possível evidenciar alguns pontos que necessitam ser considerados em um modelo de avaliação, com base neste aspecto foram pesquisados e avaliados alguns modelos para avaliação de linguagens multiplataforma. Entre os modelos avaliados na revisão sistemática o trabalho *Towards the definitive evaluation framework for cross-plataform app development approaches,* apresentado por RIEGER; MAJCHRZAK (2019) foi o trabalho que mais se aproximou do modelo de avaliação de linguagem necessário.

Este trabalho contém diferentes perspectivas de avaliação separados em quatro, infraestrutura, desenvolvimento, aplicação e uso. Esses quatro pontos contêm subtópicos que abarcam todas as avaliações necessárias, além de trazer mais pontos de avaliação. O trabalho traz também a possibilidade de adicionar pesos a cada um dos pontos de avaliação, dando a possibilidade de modificar o modelo de acordo com a necessidade específica do ambiente a ser avaliado.

Este trabalho foi escrito no início de 2020, e foi baseado em diferentes modelos e trabalhos que trazem avaliações de aplicações multiplataforma, trazendo assim o estado da arte nesses ambientes além de trazer um embasamento teórico robusto.

5.1 Base teórica

RIEGER; MAJCHRZAK (2019) apresentaram em seu estudo, trabalhos que avaliassem linguagens multiplataforma, realizando uma pesquisa em bases de conhecimento científico.

A partir dessa pesquisa mais de 30 trabalhos dos últimos 5 anos foram escolhidos, esses trabalhos foram a base pela qual todos os pontos de avaliação formam escolhidos e descritos, fundamentando assim cada ponto de avaliação do modelo em trabalhos que seguem o modelo científico e tem comprovação teórica.

5.2 Multiplataforma além do *mobile*

Como pode ser observado, até pouco tempo os ambientes de desenvolvimento de aplicativos existiam somente para *smartphones* (RIEGER; MAJCHRZAK, 2019), aspecto que tornou-se praxe ao se associar o desenvolvimento multiplataforma a

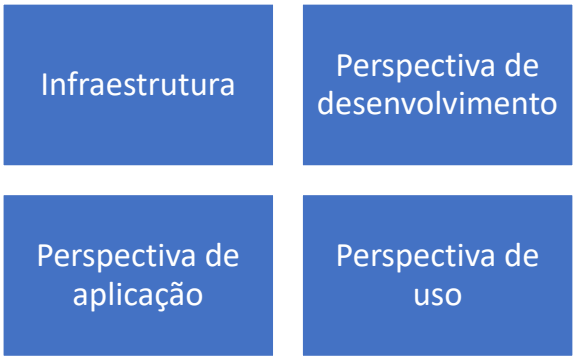
ambientes *mobile*, no entrando com o crescimento de equipamentos ligados ao ambiente de IoT, onde cada vez mais estes ambientes se apresentam como passiveis de utilizar aplicativos(NANJAPPAN *et al.*, 2017), sendo assim é possível perceber que existe a necessidade de que *frameworks* de desenvolvimento multiplataforma também considerem as características destes ambientes.

Neste *framework* de avaliação, foi considerado desenvolvimento em ambientes não *mobile*s como parte da avaliação, e como parte do modelo de pesos que apresenta o resultado da aplicação, podendo o ambiente mudar a avaliação de uma linguagem no mesmo tópico, já que um tópico como segurança podem ser avaliados de maneira diferente de um celular para um sistema de controle de um carro.

5.3 Critérios

No modelo múltiplos critérios são apresentados, dando a possibilidade de escolher somente os mais importantes para a avaliação que será feita, esses critérios são divididos em quatro categorias: infraestrutura, perspectiva de desenvolvimento, perspectiva de aplicação e perspectiva de uso assim como demonstrado na Figura 6.

Figura 6 – Figura 6 - Categorias de Avaliação



Fonte: DO AUTOR (2020)

Nos próximos subcapítulos serão apresentados em detalhes as quatro categorias para análise.

5.3.1 Modelo de Avaliação de Infraestrutura

A categoria infraestrutura está ligada a características que precisam ser previamente preparadas como licença, suporte, canais de distribuição etc. A linguagem escolhida precisa ser avaliada nessa categoria considerando os critérios de:

- Licença: Restrições no uso e distribuição de aplicações pode acontecer sem as devidas preocupações com a licença advinda da linguagem.
- Suporte a plataforma em que se planeja desenvolver: Com os ambientes escolhidos é importante considerar se a linguagem suporta a plataforma escolhida.
- Suporte a plataformas de desenvolvimento: As ferramentas necessárias para o desenvolvimento para a linguagem escolhida estão facilmente disponíveis nos ambientes escolhidos.
- Canais de distribuição: Em que *stores* de aplicativos a aplicação desenvolvida por aquela linguagem pode ser distribuída.
- Monetização: Modelos de Monetização disponíveis.
- Internacionalização: Em que países pode haver restrição para aquela linguagem.
- Suporte a longo prazo: Verificar se a linguagem presta suporte a longo prazo.

5.3.2 Modelo de Avaliação de desenvolvimento

A linguagem escolhida precisa ser avaliada referente a visão de desenvolvimento, visão essa que está ligada a capacidade do desenvolvimento em si, afinal um *framework* multiplataforma só é valido se for possível desenvolver nele. Apresentando os seguintes critérios:

- Ambiente de desenvolvimento: Verificar se existe integração com IDE’s conhecidas.
- Tempo de preparação: Quanto tempo será necessário para adaptar a equipe a linguagem.

- Escalabilidade: Como é o suporte a divisão de trabalho em um projeto por múltiplos desenvolvedores.
- Compatibilidade com modelo de desenvolvimento usado: Da modelo cascata até o ágil, a linguagem é compatível com o modelo usado?
- Design de interface de usuário: Como é o modelo de desenvolvimento para a interface de usuário.
- Teste: Existe suporte para automação de testes?
- Entrega contínua: Como a linguagem se comporta com a entrega contínua?
- Gerenciamento de configurações: Como a linguagem suporta diferentes versões do mesmo aplicativo nativamente?
- Manutenibilidade: Como é a manutenção de código na linguagem.
- Funcionalidade: Se necessário criar um aplicativo mais robusto, como a linguagem se comporta?
- Integração de código customizado: Havendo necessidade de integrar outra linguagem na aplicação, como a linguagem se comporta?
- Tempo de desenvolvimento: Como a linguagem afeta o tempo de entrega de desenvolvimento?

5.3.3 Modelo de Avaliação de aplicação

Como a aplicação se comporta em relação ao suporte de capacidades nativas, como acesso a hardware e funcionamento do ambiente físico. Contém os seguintes critérios:

- Acesso a driver específico de hardware: Acesso a câmera, giroscópio etc.
- Acesso a funcionalidade específica da plataforma: Algumas aplicações ou funcionalidades podem ser especificas de um ambiente, como a linguagem reage a elas?
- Suporte a dispositivos conectados: Como a linguagem reage a outros dispositivos conectados no ambiente.
- Igualdade de entradas e saídas: Existem várias entradas e saídas de dados diferentes nos ambientes atuais, a linguagem tem um padrão para tratar essas entradas e saídas?

- Ciclo de vida da aplicação: Ciclo que um app tem no uso diário como pausar, sair entrar entre outros.
- Segurança: Acesso indevido de dados durante uso da aplicação.
- Robustez: A linguagem tem tolerância a erros forte?

5.3.4 Modelo de Avaliação de Uso da Aplicação

A avaliação das perspectivas de uso da aplicação em relação ao usuário final, e traz consigo os critérios:

- Elementos de interface de usuário: Como os elementos interagem a partir da avaliação de quem os usa na interface?
- Performance: Qual a performance de uma aplicação desenvolvida na linguagem.
- Padrões de uso: Muito da experiencia de um usuário vem do conforto com os padrões de uso que ele está habituado, como a linguagem se adapta a esses padrões?
- Autenticação de usuário: Como a linguagem trata controle de usuários?

5.4 Questionário de Avaliação da Linguagem *Flutter* e Nativo

Durante o estudo do trabalho de RIEGER; MAJCHRZAK (2019) foi verificado que alguns dos pontos das quatro características precisavam de avaliação referente a opinião dos desenvolvedores e usuários. Em razão da falta de público e verba disponível para a avaliação com usuários, foi realizada uma pesquisa com desenvolvedores da área com experiência em Android e iOS.

Ao preparar o questionário para avaliar a opinião dos desenvolvedores, foram separadas dezesseis perguntas, baseadas em pontos específicos da avaliação como os” Elementos de interface de usuário”, presente no “Modelo de Avaliação de Uso da Aplicação”.

Para preparar a pesquisa foi analisada a pesquisa aplicada no NuBank (FREIRE *et al.*, 2019), em conjunto com pontos de avaliações de outros trabalhos como o de RIEGER; MAJCHRZAK (2019) e SHAH; SINHA; MISHRA (2019). Com esses trabalhos e questionários como base foram formuladas as seguintes perguntas:

Tabela 2 - **Perguntas do questionário dos desenvolvedores**

	Perguntas do questionário realizado com desenvolvedores Nativo/*Flutter*
1	Quantos anos de Experiência com Programação Android/iOS nativo você tem?
2	Quantos anos de Experiência com Programação em *Flutter* você tem?
3	Você tem se sentido confortável usando *Flutter*?
4	*Flutter* tem uma boa integração com a IDE utilizada?
5	As mensagens de erro têm ajudado a resolver os problemas?
6	Foi necessário adicionar camadas extras de segurança a aplicação por estar usando *Flutter*?
7	Houve muita dificuldade inicial para entender a estrutura lógica (loops, variáveis e funcionamento do código)?
8	Você enfrentou algum problema em *Flutter* que não foi possível resolver online?
9	Como você se sente em relação a curva de aprendizado em *Flutter*?
10	A Ui possui aspecto similar ao nativo, se não tem, é possível imita-lo?
11	A interação com aplicações feitas em *Flutter* como barras de navegação, rolagem, e gestos possui aspecto similar ao nativo, se não tem, é possível imita-lo?
12	Como tem sido a experiência em *Flutter* em relação à aplicações nativas no tempo de carregamento inicial da aplicação?
13	Como tem sido a experiência em *Flutter* em relação a aplicações nativas na velocidade da aplicação para alterar as *views*?
14	Como tem sido a experiência em *Flutter* em relação a aplicações nativas nos cálculos resultantes da interação do utilizador?
15	Como tem sido a experiência em *Flutter* em relação a aplicações nativas na percepção da velocidade de acesso à rede?
16	Como tem sido a experiência em *Flutter* em relação a aplicações nativas na percepção de estabilidade?

Fonte: DO AUTOR (2020)

O questionário completo se encontra no apêndice A deste trabalho. A mecânica da avaliação ocorreu da seguinte forma: inicialmente, se entrou em contato com grupos de desenvolvedores voltados a linguagens multiplataforma, nestes grupos: Flutterando(FLUTTERANDO, 2019), Papo de dev(DEV, 2019) com o total de 3000 membros e comunidades de desenvolvimento Android no Medium. Nessas comunidades de desenvolvedores que já tinham sido apresentadas a plataforma foi feito o questionamento por meio de perguntas, com o contato do Telegram disponibilizado para dúvidas. A pesquisa ficou disponível por duas semanas e foi reencaminhada diariamente nestes grupos, havendo assim a liberdade dos desenvolvedores respondê-la.

Depois de transcorridas as duas semanas, foram contabilizadas dezessete respostas, os dados foram planilhados conforme segue na Tabela 2 e foram criados gráficos a partir das respostas que serão apresentados no decorrer desta pesquisa.

Tabela 3 - Resultado da pesquisa com desenvolvedores

	Respostas				
Perguntas	a	b	c	d	e
Pergunta 1	8	7	2	-	-
Pergunta 2	9	8	0	-	-
Pergunta 3	0	0	1	6	10
Pergunta 4	0	7	10	-	-
Pergunta 5	1	7	9	-	-
Pergunta 6	1	16	-	-	-
Pergunta 7	7	10	0	-	-
Pergunta 8	5	12	-	-	-
Pergunta 9	0	3	3	9	2
Pergunta 10	11	0	6	-	-
Pergunta 11	11	0	6	-	-
Pergunta 12	1	11	2	-	-
Pergunta 13	1	7	9	-	-
Pergunta 14	0	12	5	-	-
Pergunta 15	0	15	2	-	-
Pergunta 16	2	13	2	-	-

Fonte: DO AUTOR (2020)

5.5 Avaliação de Critérios

Para a Aplicação da avaliação foram realizadas pesquisas sobre cada um dos modelos de avaliação apresentados anteriormente neste capítulo, tanto para *Flutter* como para nativo. Os pontos em que o trabalho de RIEGER; MAJCHRZAK (2019) levanta a possibilidade de realizar questionamento direto aos desenvolvedores também serão apresentados, considerando as respostas como apresentado anteriormente.

Além da pesquisa e do questionário foi feito um protótipo de uma aplicação que foi desenvolvida em ambiente nativo, utilizando sua interface para análise, porém foi

refeita em *Flutter*, e tem alguns pontos apresentados na avaliação. Com essas informações então foi apresentada a conclusão da pesquisa.

6 RESULTADO DA APLICAÇÃO DA PESQUISA

Com base no modelo descrito no capítulo 5 é necessário avaliar *Flutter* considerando uma categorização específica. Neste capítulo será apresentado o resultado da pesquisa em relação a avaliação das quatro categorias. Além da pesquisa sobre os conceitos apresentados anteriormente, também será apresentado os resultados do questionário com os desenvolvedores.

6.1 Avaliação sobre Infraestrutura

A preparação na escolha de um *framework* ou linguagem multiplataforma passa pelo planejamento de alguns pré-requisitos em seu ciclo de desenvolvimento, para tanto, a pesquisa realizada necessitou se basear em pesquisa sobre a linguagem e fontes já estabelecidas, não havendo simulação executada.

Inicialmente é importante perceber se é possível realizar o desenvolvimento do projeto na linguagem que se escolheu, nem todas as linguagens possuem liberdade completa de distribuição após o desenvolvimento, isso e controlado pela licença.

Flutter está sobre a licença BSD, o modelo BSD pode ser usado tanto como modelo, quanto como licença, e tem por característica ser permissiva. Isso significa que a licença não afeta programas feitos em *Flutter* a não ser quando especificado o contrário, podendo assim ser utilizado qualquer outo modelo de licença para programas feitos em *Flutter* sem que a licença do próprio *Flutter* interfira(THE FREEBSD PROJECT, 1992), (JIM, SIMON *et al.*, 2020).

O kit de desenvolvimento de software do Android está sobre uma licença própria, com regras especificas de desenvolvimento e distribuição, podendo ser lida por completo no link https://developer.android.com/studio/terms?hl=pt-br. Essa licença proíbe desenvolvimento de aplicações para outras plataformas fora do que pode ser encontrado no site de compatibilidade do Android encontrado no link https://source.android.com/compatibility.

O kit de desenvolvimento da Apple permite somente utilização com ambientes homologados pela Apple como é possível verificar em sua licença de desenvolvimento encontrada no link https://developer.apple.com/terms/apple-developer-agreement/Apple-Developer-Agreement-Portuguese-Brazilian.pdf, assim como

restringir a qualquer momento o acesso a aplicação ao seu ambiente, por regras existentes após a criação da aplicação ou anteriores a ela.

Uma das características em que o *Flutter* mais se destaca é o seu suporte amplo às muitas plataformas existentes. Atualmente as linguagens suportadas oficialmente são Android, IOs e Linux, suporte beta a Web e alpha a Windows e MacOS(JIM, SIMON *et al.*, 2020).

Como é possível perceber o suporte a diferentes plataformas está em diferentes estágios de desenvolvimento. Atualmente *Flutter* tem suporte pleno a Versão 4.1.x no Android e superiores, e versão 8 e superiores para IOs. Considerando a parte de hardware *mobile* dos ambientes *Flutter* suporta iPhone 4S ou mais novos e dispositivos Android ARM.

Para aplicações Web o navegador suportado é o Chrome. Para aplicações Desktop o desenvolvimento em Linux é mais avançado, com a canonical oferecendo suporte oficial ao desenvolvimento em *Flutter* para seu sistema operacional(CANONICAL, 2020). Em relação a Windows e MacOS, por mais que já seja possível desenvolver nesses ambientes eles estão em suporte alpha somente.

As linguagens nativas suportam somente o ambiente padrão, sendo o Java para desenvolvimento Android suportando apenas Android e o Xcode para iOS apenas o ambiente do iOS(GOGGLE, 2020), (SR. IOS DEVELOPER, 2018).

Como *Flutter* compila o código desenvolvido em nativo, é possível incluir seus programas compilados nas Stores oficiais de cada sistema operacional, assim como algumas stores já prestam suporte oficial como a canonical com a *Snap Store* (CANONICAL, 2020), *appStore* do Apple, *Playstore* da Google e Huawei *AppGallery* da Huawei(HUAWEI, 2020), já as aplicações nativas suportam somente as lojas que existem em seus ambientes.

Como a licença do *framework* é baseada na licença BSD, qualquer modelo legal de monetização é aceito em programas criados em *Flutter*, sendo possível se utilizar aplicativos premium, anúncios, freemium, comércio eletrônico, assinatura, compras no aplicativo (IAP) e modelos híbridos dos modelos anteriores, desde que esteja em acordo com as regras da respectiva store (APPLE, 2020b), (JIM, SIMON *et al.*, 2020), (THE FREEBSD PROJECT, 1992), (GOOGLE, 2020). Aplicações nativas seguem as mesmas regras em monetização, dependendo das regras impostas pela *store* de distribuição.

Qualquer aplicação, por mais que seja desenvolvida seguindo um mesmo padrão, pode sofrer alterações dependendo do país onde se encontra. Tanto leis como condições únicas podem afetar os parâmetros de uma aplicação no país onde ela será distribuída. É importante frisar que, independentemente da licença isso pode acontecer (THE FREEBSD PROJECT, 1992).

Um exemplo disso são aplicações com regras especificas para o ambiente chinês que estão descritas no documento intitulado “A Secretaria Estadual de Serviços de Informação da Internet emitiu o Regulamento sobre a Administração de Serviços de Informação para Aplicativos de Internet”(tradução automática)(CHINA, 2016).

Em razão dessas diferenças a linguagem que se for desenvolver normalmente traz a possibilidade de alterar configurações da aplicação de acordo com a região onde será instalado. Tanto *Flutter* como as linguagens nativas possuem essa função, que dá assim menor tempo em manutenção de código para o desenvolvedor (JIM, SIMON *et al.*, 2020), (JIM, SIMON *et al.*, 2020), (SR. IOS DEVELOPER, 2018).

Além do suporte oficial da Google em relação a problemas no *framework*, é possível adquirir suporte da comunidade, já que sendo *Open Source* e possuindo a licença BSD a linguagem pode ser alterada pela própria comunidade.

Essa característica tem sido considerada importante para o suporte a longo prazo pois, mesmo se a empresa criadora e mantenedora do *framework* deixar de prestar suporte a comunidade tem total liberdade técnica e legal de assumir o projeto, podendo em teoria o manter indefinidamente(JIM, SIMON *et al.*, 2020), (THE FREEBSD PROJECT, 1992), (OZGUL *et al.*, 2010).

Após pesquisa na documentação do *SDK* do Android pelos termos, “suporte”,” atualização”,” support”,” lifecicle” e ”end of life”, e nenhum dos resultados das pesquisas desses termos trazia a informação sobre o tempo de suporte da mesma, nos blogs de segurança do time do Android é possível verificar que a versão do Android recebe atualização por dois anos e atualizações de segurança por três anos (ANDROID TEAM, 2015).

Como o sistema Android nativo é baseado em Java, foram pesquisados os períodos de suporte ao desenvolvimento Java e o período de suporte, e foi possível verificar que os suportes “premium” têm uma data de fim, como demonstrado na Tabela 4:

Tabela 4 - **Oracle Java SE Roadmap de Suporte**

Oracle Java SE Roadmap de Suporte				
Versão	**Lançamento**	**Suporte Premier**	**Suporte Estendido**	**Suporte de Manutenção**
7	jul/11	jul/19	jul/22	Indefinido
8	mar/14	mar/22	dez/30	Indefinido
9 (não-LTS)	set/17	mar/18	Indisponível	Indefinido
10 (não -LTS)	mar/18	set/18	Indisponível	Indefinido
11 (LTS)	set/18	set/23	set/26	Indefinido
12 (não -LTS)	mar/19	set/19	Indisponível	Indefinido
13 (não -LTS)	set/19	mar/20	Indisponível	Indefinido
14 (não -LTS)	mar/20	set/20	Indisponível	Indefinido
15 (não -LTS)	set/20	mar/21	Indisponível	Indefinido

ORACLE (2020)

Em relação a iOS por mais que a Apple preste suporte de 7 anos para seus equipamentos não há um tempo específico de suporte para uma versão iOS, com as variações sendo apresentadas e atualizadas constantemente(APPLE, 2020a, c).

A canonical tem um suporte de 5 anos para suas versões LTS, porem o desenvolvimento envolve múltiplas linguagens com licenças e suportes diferentes(CANONICAL, 2015).

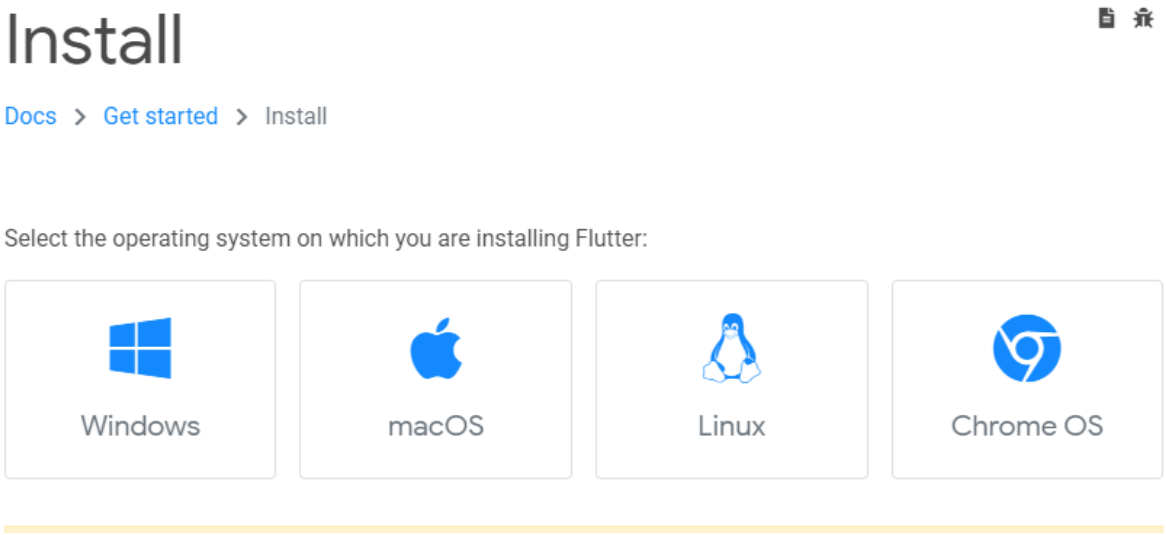
6.2 Avaliação sobre Desenvolvimento

O desenvolvimento de uma aplicação envolve muitos fatores, utilizando a pesquisa de Rieger (RIEGER; MAJCHRZAK, 2019) foram pesquisados pontos de avaliação citados no capitulo 5.

Para se iniciar o desenvolvimento em qualquer linguagem, normalmente é necessário antes instalar o kit de desenvolvimento ligado àquela linguagem. Em *Flutter* essa necessidade também existe e a documentação do *Flutter* contém o passo a passo de como realizar esta ação(JIM, SIMON *et al.*, 2020).

Na documentação oficial do *Flutter* é possível verificar que o ambiente em que se deseja desenvolver pode alterar o modelo de instalação, atualmente os ambientes suportados são: Windows, macOS, Linux e Chrome OS como demonstrado na Figura 7, que foi retirada da documentação de instalação do site de documentação do *Flutter*.

Figura 7 - **Ambientes de instalação do SDK do *Flutter***



Fonte: JIM, SIMON (2020)

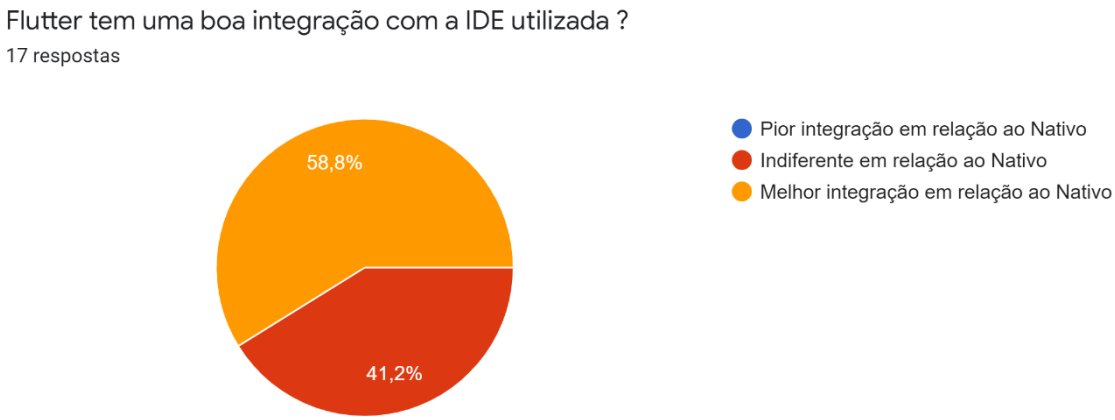
Para realizar a instalação no Windows é necessário possuir a versão Windows 7 SP1 ou superiores, é necessário possuir o Git para Windows instalado. Então é necessário realizar o download da SDK e modificar o path do Windows para que o *Flutter* seja reconhecido em qualquer janela. Após a instalação do SDK do *Flutter* é necessário seguir os mesmos passos que se seguiria na instalação dos toolkits nativos de cada linguagem. Sendo assim instalado o *Flutter* ainda será necessário instalar os SDK's das linguagens nativas (JIM, SIMON *et al.*, 2020).

Por mais que seja possível instalar o SDK do Android sem o Android Studio, todo guia oficial do Google pedirá para que seja instalado o Android Studio e então realizada a configuração automática do SDK do Android pelo Android Studio (GOGGLE, 2020). O Xcode, que contém o SDK do iOS pode ser instalado a partir da Mac App Store(APPLE INC., 2016).

Outro ponto importante durante o desenvolvimento é a utilização da IDE, que fornecerá o ambiente de trabalho do desenvolvedor, em razão disso no questionário aplicado com os desenvolvedores foi questionado se a IDE tinha uma boa integração com *Flutter*, comparando a integração atual do nativo com a IDE que este utilizava.

No resultado é possível perceber que nenhum desenvolver considerou a integração com a IDE pior do que a integração que a linguem nativa possui, e mais da metade consideraram melhor a integração da IDE com i, como demonstrado no Gráfico 2

Gráfico 2 - Ambientes de instalação do SDK do *Flutter*



Fonte: DO AUTOR (2020)

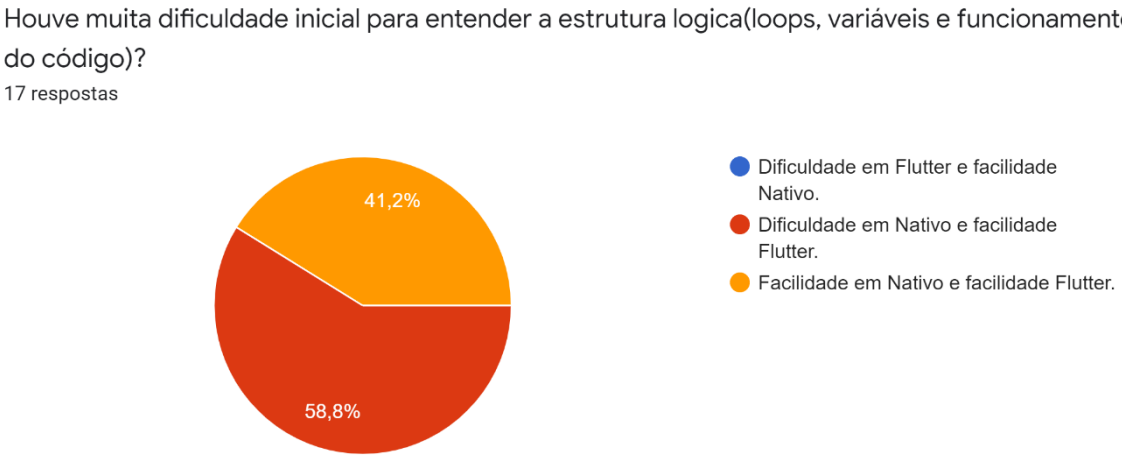
Há também um tempo necessário para a equipe se adaptar ao uso da nova linguagem, nesse caso *Flutter*. Como o tempo pode ser relativo de equipe para equipe e não houve verba e tempo para uma pesquisa completa neste ponto, e para poder haver maior confiabilidade foi realizado uma busca por trabalhos acadêmicos que tenham estudado o assunto.

Foi realizado uma busca em agregadores de trabalhos acadêmicos como o Google Schoolar, na biblioteca da Feevale, assim como agregadores padrão, pelos termos “tempo de aprendizado *flutter*”,”tempo de adaptação *flutter*”, “*flutter*,equipes,aprendizado”, e não foram encontradas pesquisas comparando o aprendizado de equipes que possuam por padrão nativo e estão mudando para *Flutter* disponíveis.

No entanto blogs do próprio Nubank indicam a facilidade com a experiência de mudança tem ocorrido, não disponibilizando no entanto suas experiências de maneira completa(FREIRE *et al.*, 2019). Em relação à pesquisa aplicada aos desenvolvedores foram feitos os questionamentos abordando o assunto.

O primeiro questionamento feito foi voltado a como foi a compreensão, em momento inicial, das estruturas lógicas do *Flutter*. Como apresentado no Gráfico 3 a maioria dos desenvolvedores teve facilidade em compreender a estrutura lógica do *Flutter*. É importante frisar que nenhum dos desenvolvedores sentiu dificuldade em compreender *Flutter* comparado a linguagens nativas.

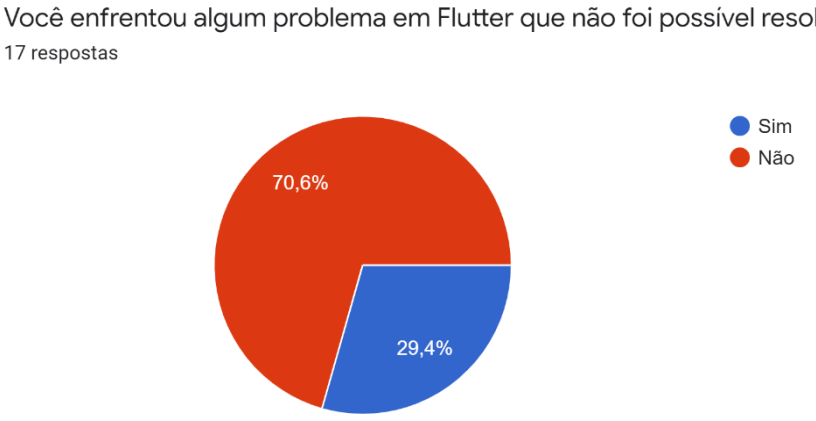
Gráfico 3 - Dificuldade inicial para entender a estrutura lógica



Fonte: DO AUTOR (2020)

Para complementar a pesquisa sobre o entendimento é necessário compreender a importância de que pesquisas online tem durante o aprendizado de uma linguagem (NASEHI *et al.*, 2012), com isso em mente foi perguntado se encontraram alguma dificuldade que pesquisas online não puderam resolver, lembrando que pesquisas online se referem a artigos já escritos e comunidades existentes. Como demonstrado no Gráfico 4 é possível perceber que 29,4% dos pesquisados possuiu algum problema que não foi possível resolver com pesquisas online.

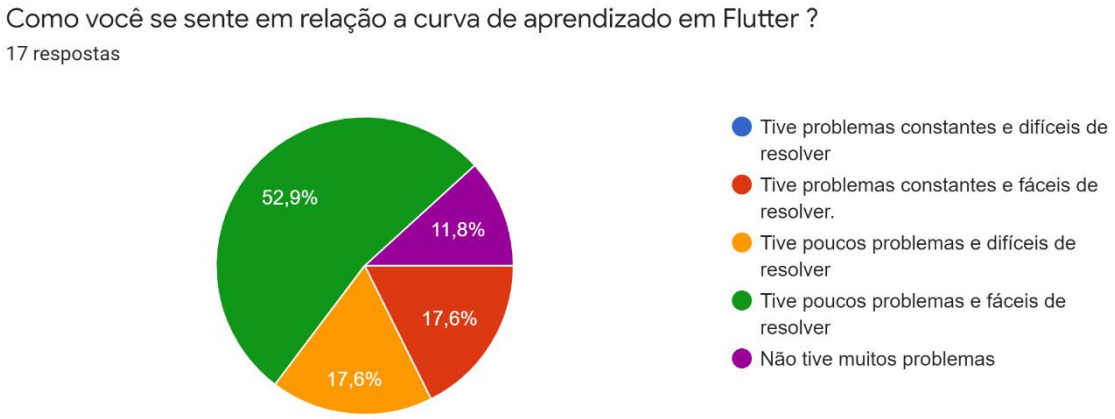
Gráfico 4 - Problemas em *Flutter* que não foram possíveis de resolver online



Fonte: DO AUTOR (2020)

Além dessas informações, também foi verificado no questionário com os desenvolvedores a dificuldade em relação a curva de aprendizado deles relativo à linguagem *Flutter*, 52,9% tiveram facilidade durante seu aprendizado, tendo poucos problemas e fáceis de resolver, e cerca de 39,4% tiveram problemas difíceis, sendo menos da metade disso problemas constantes e difíceis como demonstrado no gráfico 5.

Gráfico 5 - Curva de aprendizado em *Flutter*



Fonte: DO AUTOR (2020)

Tendo como base o resultado do questionário aplicado aos desenvolvedores, por mais que houveram problemas durante o aprendizado da linguagem, mais da metade apresenta disposição a uso do *Flutter* em relação ao Nativo.

É interessante reparar, no entanto, que não foi encontrada pesquisa acadêmica em relação ao assunto em específico. Por mais que a pesquisa com os desenvolvedores apresente uma preferência por *Flutter,* não foi pesquisado ou aplicada nenhuma metodologia de estudo para esse caso em específico, e como este trabalho aborda uma visão geral, não será feita uma pesquisa de metodologia específica para este ponto, podendo esta ser feita em trabalho futuro.

Para a constância do trabalho de desenvolvimento de uma linguagem ser aplicada, é necessário compreender como esta pode ser desenvolvida em conjunto com outros desenvolvedores. Atualmente a função de *hot reload*(capacidade de alterar um código, e sem compilar o código inteiro novamente, aplicar correções) nativa do *Flutter* aumenta a capacidade de escalabilidade de desenvolvimento, além

disso *Flutter* tem ferramentas de *debbug* de código e de interface nativamente(FREIRE *et al.*, 2019), (JIM, SIMON *et al.*, 2020).

Também é necessário compreender que as metodologias ágeis tem um grande papel nos modelos de desenvolvimentos atuais (LÖFFLER; GÜLDALI; GEISEN, 2010), em razão disso é necessário verificar o tempo necessário para a equipe se adaptar ao uso da nova linguagem, aplicando a metodologia. Como o tempo pode ser relativo de equipe para equipe e não houve verba e tempo para uma pesquisa completa neste ponto, e para poder haver maior confiabilidade foi realizado uma busca por trabalhos acadêmicos que tenham estudado o assunto.

Foi pesquisado termos como “scrum,*flutter*”,”tempo de adaptação *Flutter* no scrum”, “*flutter*,equipes,scrum”, “metodologias ágeis e *flutter*”, tanto em navegadores e buscadores padrão quanto em buscadores voltados ao mundo acadêmico e bibliotecas digitais como a biblioteca da Feevale e não foi encontrado pesquisas acadêmicas que demonstrasse alguma diferença entre uso de *Flutter* ágeis disponíveis. Nesse ponto seria necessário realizar um trabalho próprio sobre esse aspecto, o que não poderá ser realizado nesse trabalho em razão do período disponível para a pesquisa.

Foi possível perceber uma diferença no desenvolvimento da interface do usuário em *Flutter*. Ao contrário das aplicações nativas em que é necessário construir as abstrações de todos os itens um a um, *Flutter* traz pré-pronto todas as abstrações nas formas de *widgets*. Com isso a facilidade de se construir uma interface baseada em abstrações já feitas e maior.

No entanto é importante perceber que em razão de se utilizar especificamente widgets construídos previamente novas abstrações não estarão disponíveis para o *Flutter* no lançamento. A comunidade precisará disponibilizar widgets ou aguardar a Google disponibilizá-los(FREIRE *et al.*, 2019), (JIM, SIMON *et al.*, 2020), (TRAN, 2020).

Um dos grandes pontos em que a atenção dada ao *framework Flutter* e sua modelação nativa para automação de teste(FREIRE *et al.*, 2019), o *framework* foi pensado de maneira que seja possível realizar testes unitários com suporte nativo, de maneira que não seja necessário renderizar toda a aplicação para execução dos testes.

Com uma programação voltada a widgets, nativamente *Flutter* também apresenta automação voltada para testes de funcionamentos de widgets específicos,

trazendo testes voltados a respostas de ações ou eventos, execução de layout e instanciar widgets filhos(JIM, SIMON *et al.*, 2020), (MATYUNINA, 2020). É importante perceber que mesmo com testes de execução de layout, durante a automação ainda não há o desenho completo da aplicação, sendo somente o widget desenhado.

Testes de integração são realizados fora da aplicação, geralmente com o aplicativo em funcionamento em um ambiente controlado. Neste caso *Flutter* se comporta como uma aplicação padrão(FREIRE *et al.*, 2019), (JIM, SIMON *et al.*, 2020).

Linguagens nativas como o Android possuem também suporte a automação, no entanto os testes realizados são voltados a teste de UI(GOOGLE, 2019a). Não foi possível encontrar na documentação do iOS algum ponto de automação nativa buscando pelos termos “automação, teste”,” automação”,” teste automatizado”.

Também é possível aplicar entrega contínua com *Flutter*, na própria documentação oficial há o passo a passo de aplicar o *FASTLANE*, uma aplicação já bem estruturada e utilizada para integração contínua. Também é possível aplicar entrega contínua com linguagens nativas(FASTLANE, 2020), (JIM, SIMON *et al.*, 2020).

Como a aplicação *FASTLANE* entre outras propõe, é possível realizar entregas versionadas da aplicação, sendo de controle final da loja aplicar qualquer das versões. Aplicações nativas seguem o mesmo padrão que o *FASTLANE*, utilizando de ferramentas terceiras para gerenciar controles de versão.

Em relação a utilização de aplicações é possível encontrar aplicações de renome no mercado já utilizando *framework Flutter* como padrão. Um dos exemplos de mercado e a aplicação *Stadia*, aplicação que é utilizada para jogar por streaming, tecnologia nova, com alguma utilização que está entrando recentemente no mercado em uma versão estável. Outra aplicação estável de grande porte é o *marketplace* da Alibaba, gigante do mercado asiático que utiliza *Flutter* de maneira hibrida em sua aplicação(GOOGLE, 2019b),(ALIBABA-FLUTTER, 2019) .

O grupo técnico da Alibaba demonstrou como utilizou sua aplicação de tecnologia híbrida de nativo com *Flutter*, utilizando um plugin que torna possível utilizar *Flutter* interagindo com linguagem nativa(XUJIM, 2019).

É possível notar que tanto no desenvolvimento realizado durante este trabalho quanto nos desenvolvimentos apresentados em por empresas que estavam testando

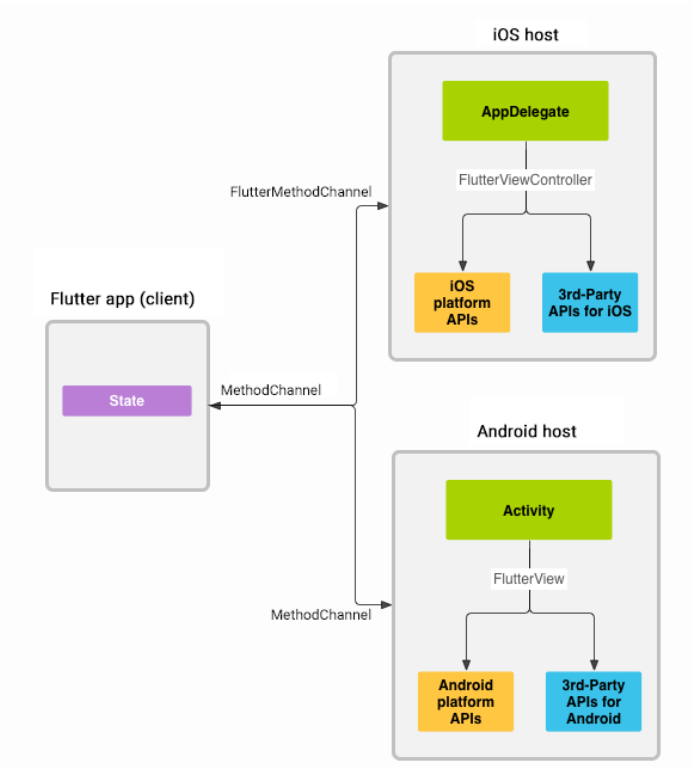
o *Flutter*, não houve diferenças grandes entre os tempos de entrega de aplicações (FREIRE *et al.*, 2019).

6.3 Avaliação sobre Características da Aplicação

As aplicações têm comportamentos de *back-end* comuns a desenvolvimento que precisam ser avaliados. É importante frisar que as linguagens nativas já possuem todas as características abaixo e, em razão disso só serão apresentadas as características da linguagem atual.

A maioria das capacidades de hardware já estão desenvolvidas para *Flutter*, no entanto é importante notar que novos dispositivos podem não estar suportados por padrão, sendo possível no entanto a própria comunidade adicionar o suporte por meio de novos plugins(JIM, SIMON *et al.*, 2020). Isso acontece pois o ambiente host ao ser atualizado disponibiliza *APIs* que atualmente já são controladas pelas *views* do *Flutter*, como demonstrado na Figura 8:

Figura 8 - Interação com api nativa ou de terceiros.



Fonte: JIM,SIMON (2020)

Pode se imaginar que assim como o hardware, poderia haver atrasos em relação ao lançamento de uma capacidade nova de software e seu possível uso no

Flutter, no entanto novas capacidades do SO podem ser usadas imediatamente após a atualização do SO *host* (JIM, SIMON *et al.*, 2020).

Assim como hardwares embarcados, a maioria das capacidades de dispositivos externos já estão desenvolvidas para *Flutter*, no entanto é importante notar que novos dispositivos podem não estar suportados por padrão, sendo possível no entanto a própria comunidade adicionar o suporte por meio de novos plugins(JIM, SIMON *et al.*, 2020).

Atualmente os ambientes de sistemas operacionais reconhecem diferentes formas de entrada e saídas de informações. Celulares atuais possuem dispositivos que medem desde sua inclinação à quantidade de luz no ambiente, e essas entradas precisam ser reconhecidas pelos programas que receberão chamadas do ambiente nativo, assim como qualquer resposta visual ou mecânica dada pela aplicação precisa estar entre as respostas disponíveis no ambiente.

Qualquer entrada e saída de dados reconhecida pelo SO ligada a software pode ser tratada por padrão com *Flutter*(JIM, SIMON *et al.*, 2020).

O ciclo seguido pelo *Flutter* é separado em dois estágios, o ciclo de vida da aplicação e o ciclo de vida dos widgets. Somente há ciclo de vida em *StatefulWidgets*, tendo os seguintes métodos(本文档使用, 2019), (JIM, SIMON *et al.*, 2020):

- *createState*: Quando se cria o *widget*, Flutter imediatamente chama esta função.
- *initState*: É o primeiro método utilizado após a criação do widget, deve ser utilizado para inicializar o estado do widget.
- *didChangeDependencies*: Esse método é chamado logo após o método *initState*, somente na primeira execução do widget.
- *Build*: Esse método é chamado logo após o método *didChangeDependencies*. Toda renderização do *widget* e feita nesse momento. Esse método é chamado sempre que há a necessidade de renderizar novamente o *widget*.
- *didUpdateWidget*: Esse método é chamado quando um widget pai é atualizado e precisa que o widget seja renderizado novamente.
- *deactivate*: Esse método é chamado quando o *Flutter* retirará o estado desse *widget* da árvore de estados.

- *dispose*: Esse método é chamado quando o objeto e seus estados são removidos da renderização de tela e não serão mais chamados.

O ciclo de vida da aplicação é apresentado a seguir:

- *inactive* – A aplicação está inativa e seu estado não está recebendo entradas de usuários. Somente iOS.
- *paused* – A aplicação não está respondendo a entradas do usuário, não está visível ao usuário e está rodando em plano de fundo.
- *resumed* – A aplicação está visível e respondendo a entrada de usuário.
- *suspending* – A aplicação está suspensa temporariamente.

Por mais que existam muitas aplicações que necessitem de níveis diferenciados de segurança, *Flutter* consegue alcançar o nível de aplicações nativas combinando plugins e API's nativas como demonstrado por RICK WU (2019). É interessante perceber, no entanto, que durante a pesquisa com os desenvolvedores essa necessidade de combinar plugins foi pouco percebida como uma camada extra como demonstrado no Gráfico 6.

Gráfico 6 - Necessidade de adicionar camadas extras de segurança usando Flutter

Fonte: DO AUTOR (2020)

Como uma linguagem em crescimento existem muitos planos de melhorias, com tratamentos obrigatórios de erros sendo implementados(podendo ser ignorados

se o programador explicitamente informar o contrário), a linguagem *DART*, que é a base de Flutter, é considerada boa para tratamento de erros(GOOGLE, 2013).

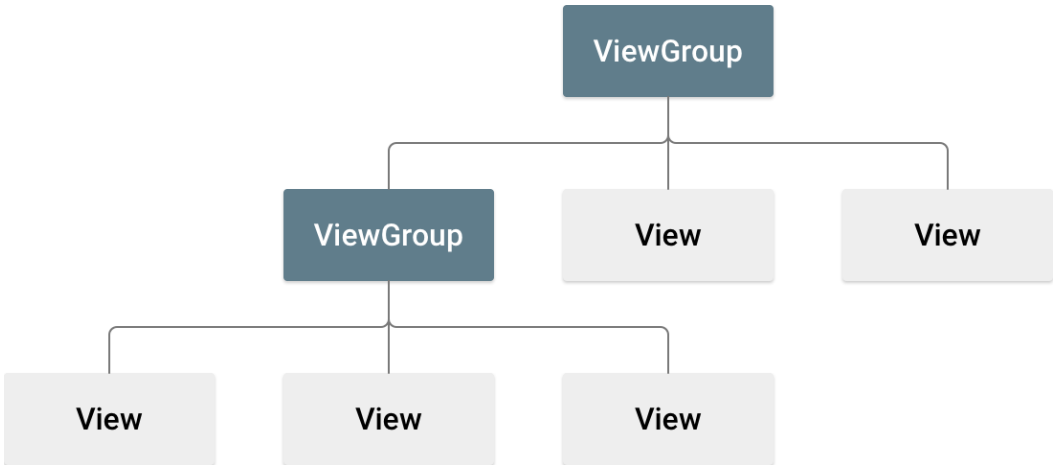
6.4 Avaliação sobre Uso da Aplicação

Durante a pesquisa sobre desenvolvimento em Flutter, foi desenvolvida uma aplicação com o objetivo de simular a utilização do usuário com base nas avaliações descritas no capítulo 5, além disso será apresentado esta simulação de desenvolvimento próprio em comparação a uma de desenvolvimento nativo.

Desenvolvimento Android nativo utiliza uma estrutura de interface de usuário hierárquica, dividida entre *View* e *Viewgroup*. *View* normalmente é algo que o usuário pode enxergar desenhado em tela ou interagir com e *Viewgroup* é um contêiner invisível que existe para estruturar o layout, tanto para objetos *View* como outros *Viewgroups*.

No Xcode é possível controlar a UI por meio de *View* e *stack*. *Views* representam parte visível da aplicação e *stacks* são utilizados para ajustes em relação a UI. Na imagem abaixo pode-se verificar o funcionamento dos layouts do Android.

Figura 9 - Ambiente de UI do Android Nativo



DEVELOPERS (2019)

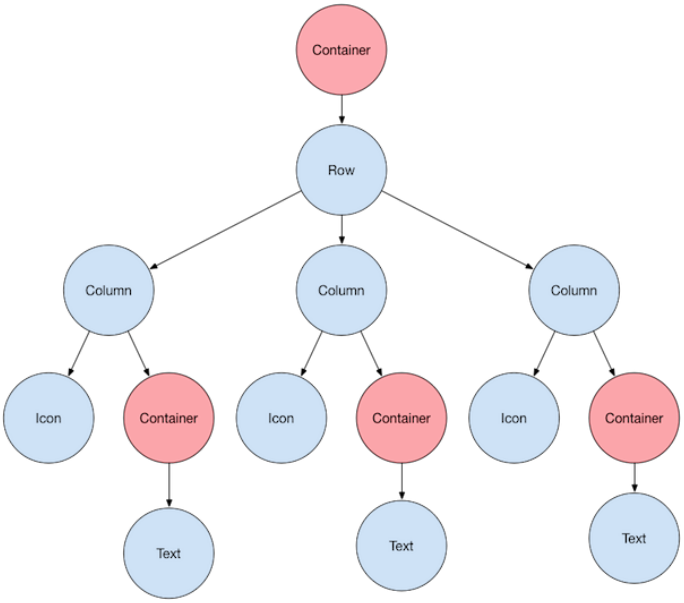
No *Flutter* o conceito de desenho de tela é diferente, como tudo foi desenhado para ser *Widgets*, todo o controle da UI é feito de maneira que cada *widget* seja feito por outros *widgets* com objetivos específicos de controle de interface. É importante destacar que como a linguagem é aberta, é possível criar *widgets* que se comportam

de maneira diferente, sendo assim não é possível descrever todos os widgets de controle de tela, no entanto alguns *widgets* serão de uso comum e podem esclarecer o funcionamento da UI por meio de *widgets.*

- Container: Este *widget* cria uma região desenhada com bordas invisíveis por padrão, com uma distância das bordas da tela e, se não conter nenhum *widget* filho, tentará se expandir para o tamanho máximo possível. É utilizado para criar regiões limitadoras que conterão outros *widgets*.
- *Row*: Este *widget* cria uma região desenhada com bordas invisíveis por padrão, com uma distância das bordas da tela e sempre desenha seus filhos em sequência de maneira horizontal.
- *Column*: Este *widget* cria uma região desenhada com bordas invisíveis por padrão, com uma distância das bordas da tela e sempre desenha seus filhos em sequência de maneira vertical.
- *Padding*: Este *widget* cria uma região desenhada com bordas invisíveis por padrão, com uma distância das bordas da tela, obrigatoriamente precisa de um widget filho e expande a área ao redor dele de acordo com a configuração utilizada.

Estes widgets demonstram como a UI em *Flutter* é desenhada durante a codificação, seguindo o exemplo da Figura 10:

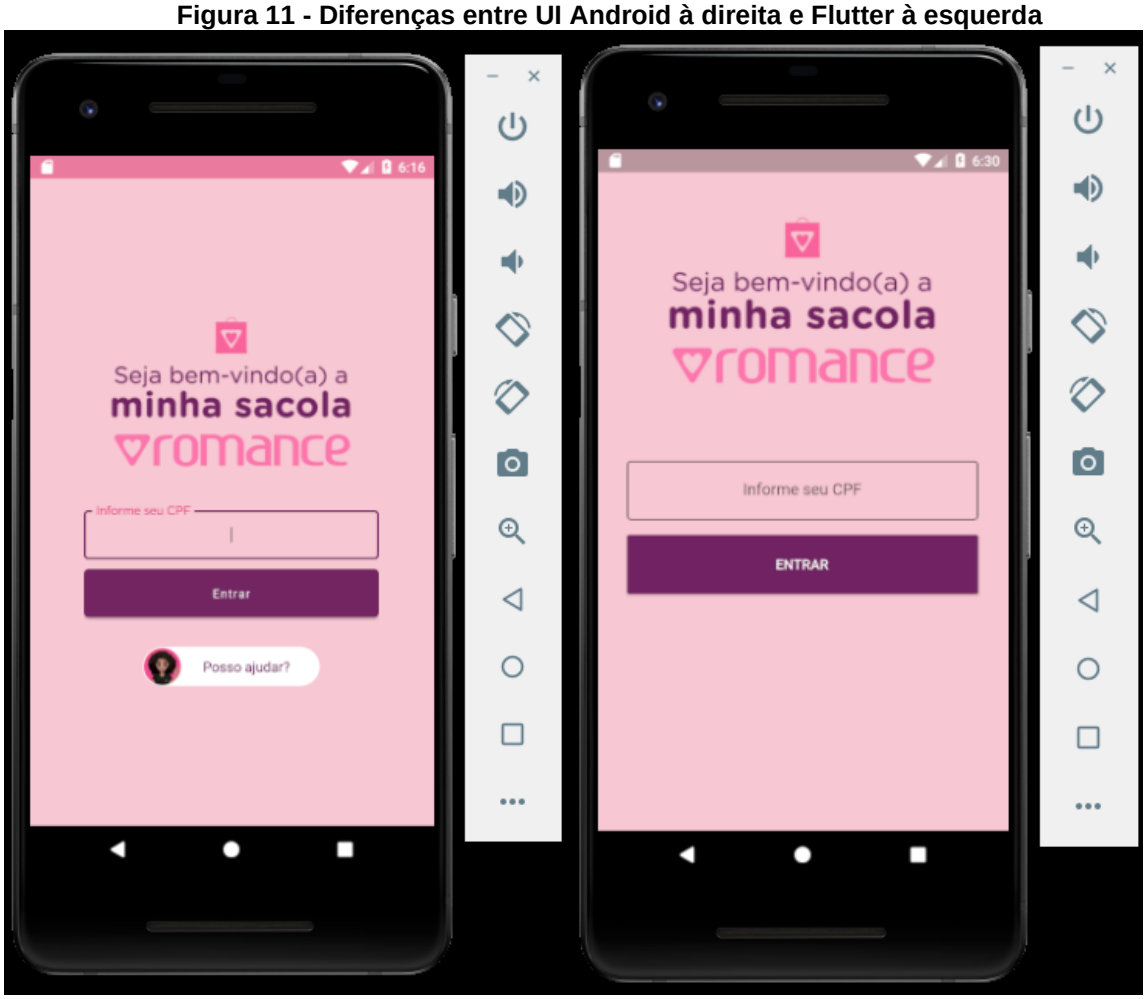
Figura 10 - Ambientes de UI do Flutter



GOOGLE LLC (2019)

Para verificar a diferença em código é possível verificar a Figura 11 contendo a tela da aplicação desenvolvida para a simulação em *Flutter* e suas versões nativas, é importante ressaltar que por mais que seja possível igualar as aplicações, questões como acesso pagos a servidores de terceiro e falta de experiência na programação impediram uma cópia exata da aplicação alvo:

Figura 11 - Diferenças entre UI Android à direita e Flutter à esquerda



Fonte: DO AUTOR (2020)

Em relação ao código, é possível observar que o modelo de desenvolvimento é diferenciado, a aplicação nativa em Android prepara um arquivo XML com as informações da UI e cria uma classe Java para controle das funções e processos. Em Flutter por mais que seja possível separar os arquivos com a UI e os processos (e muitos modelos de desenvolvimento incentivam essa prática), a criação da UI e dos processos utiliza o *framework* da mesma maneira, podendo assim ser tratados como um só processo.

No questionário com os desenvolvedores, para verificar sua percepção no tocante à similaridade e uso da UI foi questionado se a UI possuía aspecto similar ao nativo, ou se era possível replicá-lo, e nenhum desenvolvedor considerou que não era possível replicar a UI nativa como apresentado no gráfico 7.

Gráfico 7 – Comparação de aspecto similar ao nativo

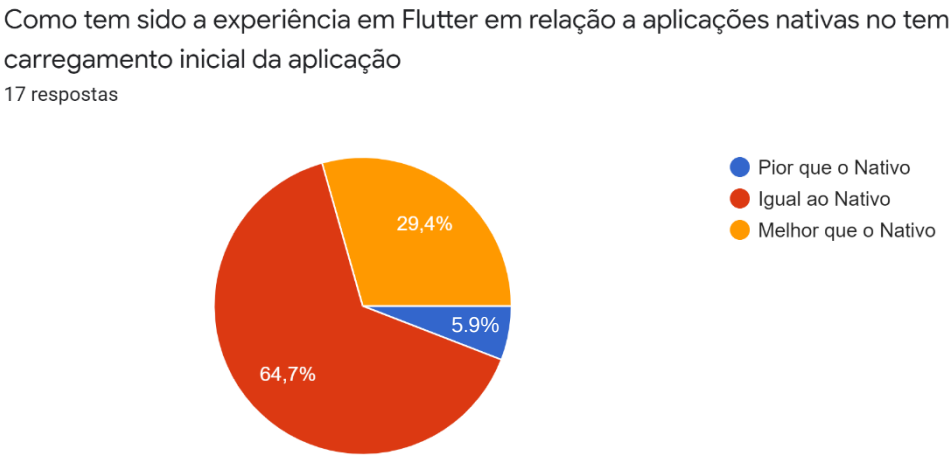
Fonte: DO AUTOR (2020)

Sendo perceptível que os desenvolvedores possuem percepção de que é possível imitar o aspecto das aplicações nativas.

Considerando a análise de WU (2018) sobre o uso da linguagem no quesito de hardware, demonstra que *Flutter* tem um desempenho melhor que outras linguagens multiplataformas em relação ao uso do hardware. Para descobrir a percepção dos desenvolvedores em relação a performance foi adicionado ao questionário perguntas sobre a percepção no tempo de carregamento e utilização de dados da aplicação.

O primeiro questionamento está ligado a percepção dos desenvolvedores em relação ao tempo de carregamento da aplicação feita em *Flutter* comparada a aplicações nativas e como apresentado no Gráfico 8, 64% dos desenvolvedores consideram que as aplicações em *Flutter* tem o carregamento inicial igual ao do nativo.

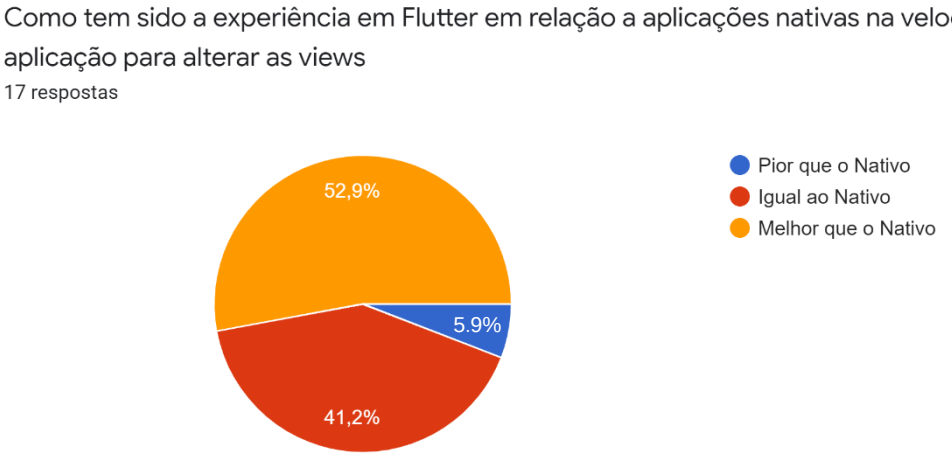
Gráfico 8 – Percepção de tempo de carregamento inicial da aplicação



Fonte: DO AUTOR (2020)

O tempo de carregamento de itens específicos na tela, as *views*, também foi questionado, pois o tempo de carregamento da tela e o tempo de carregamento de um item específico são diferentes. 52,9% dos desenvolvedores consideram melhor que o nativo, em compensação somente 5.9% consideraram pior que o nativo.

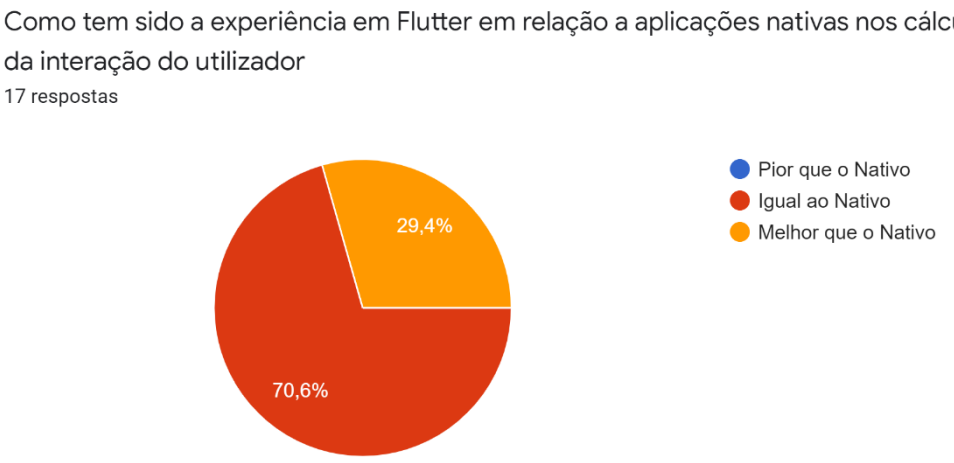
Gráfico 9 - Experiência em Flutter em relação a aplicações nativas em relação a velocidade



Fonte: DO AUTOR (2020)

Outro ponto questionado foi em relação a velocidade de resposta após interação do utilizador, considerando a resposta como resultado de algum cálculo feito pela aplicação. Nenhum dos desenvolvedores considerou que o tempo da resposta de aplicações em Flutter foi pior que o nativo, enquanto a maioria considerou igual, como apresentado no Gráfico 10.

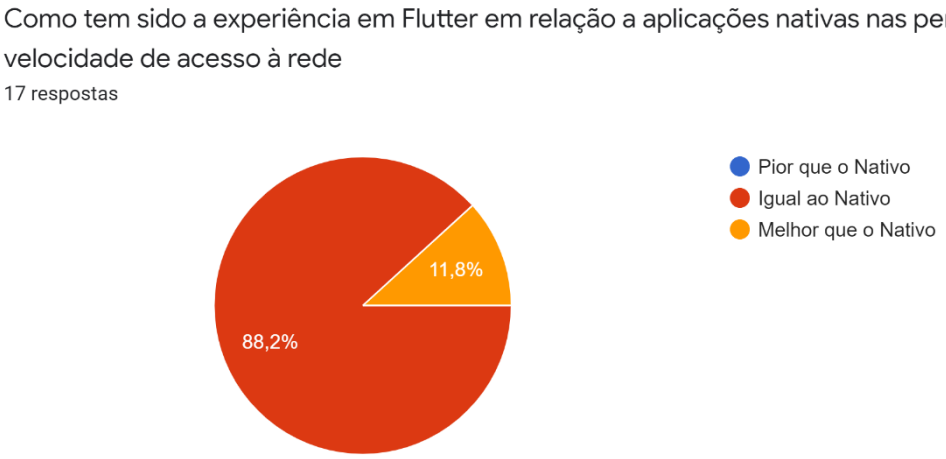
Gráfico 10 - Experiência em Flutter em relação a aplicações nativas em cálculos resultantes da interação do utilizador



Fonte: DO AUTOR (2020)

Também foi questionado sobre o tempo de resposta de chamadas de rede, com o resultado como o Gráfico anterior, com a maioria considerando igual ao nativo e nenhum considerando pior que o nativo, como apresentado no Gráfico 11.

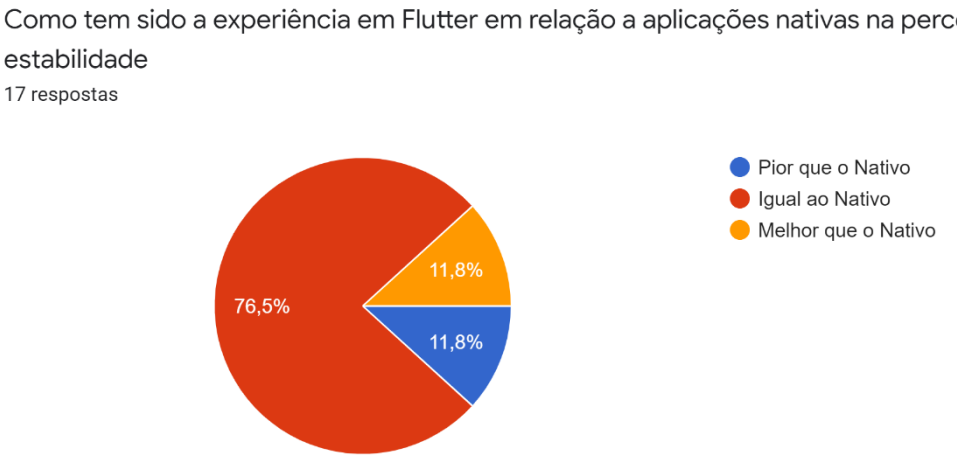
Gráfico 11 - Experiência em Flutter em relação a aplicações nativas na percepção da velocidade de acesso à rede



Fonte: DO AUTOR (2020)

Por último, foi perguntado em relação a percepção de estabilidade, com somente 11.8% considerando pior que o nativo, como apresentado no Gráfico 12.

Gráfico 12 - Experiência em Flutter em relação a aplicações nativas na percepção de estabilidade



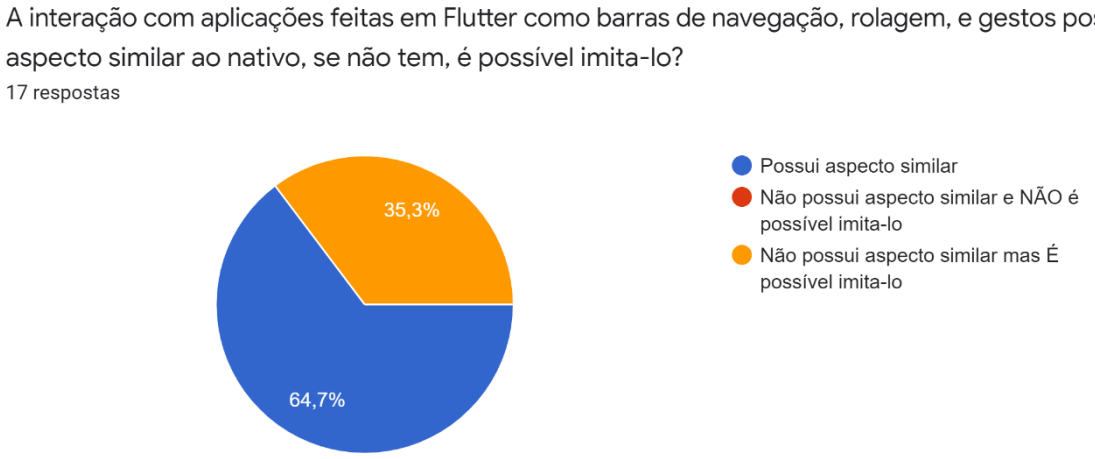
Fonte: DO AUTOR (2020)

Sendo assim é possível verificar que mais de 60% dos desenvolvedores consideram o tempo em que aplicações desenvolvidas em *Flutter* atende os padrões nativos e somente um dos entrevistados considerou o tempo pior do que aplicações nativas.

Além da velocidade, é necessário também verificar a percepção de como a aplicação flui, ou seja, se o uso da aplicação se compara a do nativo. Como informado anteriormente, a não ser que haja processos novos recém desenvolvidos, todos os outros modelos de desenvolvimento se encaixam com a utilização de widgets.

Na percepção dos desenvolvedores o *Flutter* traz essa possibilidade como demonstrado no Gráfico 13.

Gráfico 13 – Experiência em Flutter em relação a aspecto similar ao nativo



Fonte: DO AUTOR (2020)

Assim é possível perceber nas respostas dos desenvolvedores que é possível seguir os padrões de uso nativo.

O último questionamento feito por Rieger; Majchrzak (2019) é se durante o processo de autenticação do usuário a linguagem pede algo diferente de processos lógicos, e não há diferenças entre aplicar uma autenticação nativa ou em Flutter, que não seja da própria lógica da linguagem(JIM, SIMON *et al.*, 2020).

Apresentando estes trabalhos, e o resultado do questionários com os desenvolvedores é finalizada a avaliação sobre os pontos apresentados por Rieger Majchrzak (2019)

7 CONCLUSÃO

O desenvolvimento multiplataforma se torna uma realidade, assim como uma promessa de crescimento no futuro (EYGM LIMITED, 2019), buscando a compatibilidade, reuso de software entre outros (BIØRN-HANSEN; GRØNLI; GHINEA, 2019a). Nesse contexto a plataforma *Flutter* se apresenta como uma entre as que podem ser utilizadas para programar para mais de uma plataforma (MAINKAR; GIORDANO, 2019).

A partir dessa possibilidade, este trabalho buscou fixar as bases teóricas para que a seja possível analisar o *framework Flutter*, e sua aplicação para desenvolvimento em múltiplas plataformas, não somente dispositivos, mas também plataformas desktop, com seus diversos sistemas operacionais, frente ao desenvolvimento nativo. Tendo como base uma revisão sistemática, foi possível buscar *framework*s de avaliação bem como os principais aspectos relevantes para avaliação consistente. Como principal resultado da revisão foi possível apresentar em detalhes o *framework* de avaliação desenvolvido por Rieger e Majcharzak (2019), tendo este *framework* como base, foi avaliado a linguagem *Flutter,* em cada uma das características.

Utilizando a pesquisa de Rieger e Majcharzak (2019) como modelo foram avaliados os pontos em relação a linguagem Flutter, tanto por meio de pesquisa e quanto por meio do questionário dos desenvolvedores, foi possível verificar que as aplicações nativas possuem vantagens em relação a utilização do hardware que é feito para elas, podendo se utilizar previamente de inovações que ainda não possuam uma biblioteca em Flutter.

Durante a pesquisa foi verificado que Flutter já traz em ambientes *mobile* muita da estabilidade e capacidade de replicar aplicações já existentes, dando, no entanto, a capacidade de replicar a aplicação em outros ambientes sem a necessidade de refazer o código para outro ambiente. Pouco se perde em relação a percepção dos desenvolvedores na dinâmica da interação do usuário entre nativo e Flutter e pela percepção dos mesmos não há diferenças de velocidade das aplicações. Enquanto igualar a UI possa trazer um esforço extra, as vantagens apresentadas neste trabalho demonstram que Flutter é uma opção para desenvolvimento *mobile*.

Foi perceptível, no entanto que outros ambientes além do *mobile* ainda são poucos explorados em Flutter, mesmo com a recente entrada da Canonical no grupo de suporte oficial do Flutter, e da entrada do mesmo no ambiente Linux.

Durante o questionário com os desenvolvedores foi possível perceber que em sua maioria não tiveram dificuldades com o aprendizado e uso do Flutter, sendo para alguns ainda mais fácil do que a curva de aprendizado do nativo.

Esses fatores fazem de Flutter hoje um *framework* de desenvolvimento que pode ser utilizado para aplicações *mobile*, mas ainda não pode ser usado como substituto para outros ambientes, estando, no entanto, a caminho de sê-lo.

A pesquisa relacionada as equipes e seu tempo de preparação e adaptação a linguagem Flutter ainda não pode ser estudada por completo sendo necessário realizar um estudo prático com uma equipe que com experiência nativa e experiência e uso de metodologias ágeis, aplicando o uso de *Flutter* em uma aplicação que atualmente utiliza desenvolvimento nativo.

Além disso também se torna necessário em trabalhos futuros realizar um estudo em larga escala com usuários de aplicações nativas que tenham a aplicação alterada para *Flutter*, e verificar a diferença na percepção do usuário.

Será também necessário reavaliar a linguagem, uma vez que um novo ambiente como ambientes desktop sejam considerados como “disponíveis” em versão final.

REFERÊNCIAS BIBLIOGRÁFICAS

ALIBABA-FLUTTER. alibaba-flutter · GitHub. 2019. Available at: https://github.com/alibaba-flutter. Acesso em: 30 out. 2020.

ANDROID TEAM. Official Android Blog: An Update to Nexus Devices. 2015. Available at: https://android.googleblog.com/2015/08/an-update-to-nexus-devices.html. Acesso em: 5 nov. 2020.

APPLE. Apple security updates - Apple Support. 2020a. **iOS**. Available at: https://support.apple.com/en-us/HT201222. Acesso em: 5 nov. 2020.

APPLE. Buy additional app features with in-app purchases and subscriptions - Apple Support. 2020b. Available at: https://support.apple.com/en-us/HT202023. Acesso em: 28 out. 2020.

APPLE. Vintage and obsolete products - Apple Support. 2020c. Available at: https://support.apple.com/en-gb/HT201624. Acesso em: 5 nov. 2020.

APPLE INC. Xcode on the Mac App Store. 2016. Available at: https://apps.apple.com/us/app/xcode/id497799835?mt=12. Acesso em: 5 nov. 2020.

ARNOLD, Ken; GOSLING, James; HOLMES, David. **THE Java™ Programming Language, Fourth Edition**. [*S. l.*: *s. n.*], 2005.

BIØRN-HANSEN, Andreas; GRØNLI, Tor Morten; GHINEA, Gheorghita. A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. **ACM Computing Surveys**, v. 51, n. 5, 1 jan. 2019a. DOI 10.1145/3241739. Available at: https://doi.org/10.1145/3241739. Acesso em: 29 maio 2020.

BIØRN-HANSEN, Andreas; GRØNLI, Tor Morten; GHINEA, Gheorghita. Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance. **Sensors (Switzerland)**, v. 19, n. 9, 2019b. DOI 10.3390/s19092081. Available at: www.mdpi.com/journal/sensors. Acesso em: 30 maio 2020.

BIØRN-HANSEN, Andreas; GRØNLI, Tor Morten; GHINEA, Gheorghita. GitHub - andreasbhansen/sensors_journal_animations_performance. 2019c. Available at: https://github.com/andreasbhansen/sensors_journal_animations_performance. Acesso em: 3 jun. 2020.

BIØRN-HANSEN, Andreas; GRØNLI, Tor Morten; GHINEA, Gheorghita;

ALOUNEH, Sahel. An Empirical Study of Cross-Platform Mobile Development in Industry. **Wireless Communications and Mobile Computing**, v. 2019, 2019. DOI 10.1155/2019/5743892. Available at: https://doi.org/10.1155/2019/5743892. Acesso em: 6 abr. 2020.

BOUSHEHRINEJADMORADI, Nader; GANAPATHY, Vinod; NAGARAKATTE, Santosh; IFTODE, Liviu. Testing cross-platform mobile app development frameworks. 4 jan. 2016. **Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015** [...]. [*S. l.*]: Institute of Electrical and Electronics Engineers Inc., 4 jan. 2016. p. 441–451. DOI 10.1109/ASE.2015.21. Available at: http://ieeexplore.ieee.org/document/7372032/. Acesso em: 16 maio 2020.

BRACHA, G. The Dart programming language. 2015. Available at: https://books.google.com.br/books?hl=pt-BR&lr=&id=UHAlCwAAQBAJ&oi=fnd&pg=PT13&dq=dart+language&ots=Po6Le2IeIN&sig=5ByMYr7T4UWxnRG8uUjkbmy2rs4. Acesso em: 6 jun. 2020.

CANONICAL. Canonical enables Linux desktop app support with Flutter. 2020. Available at: https://ubuntu.com/blog/canonical-enables-linux-desktop-app-support-with-flutter. Acesso em: 24 out. 2020.

CANONICAL. Ubuntu Documentation. 2015. Available at: https://docs.ubuntu.com/. Acesso em: 5 nov. 2020.

CHINA, Cyberspace Administration of. A Secretaria Estadual de Serviços de Informação da Internet emitiu o Regulamento sobre a Administração de Serviços de Informação para Aplicativos de Internet movel. 2016. Available at: http://www.cac.gov.cn/2016-06/28/c_1119123114.htm. Acesso em: 4 nov. 2020.

DEV, Papo de. Papo de Dev - Página inicial | Facebook. 2019. Available at: https://www.facebook.com/papoddev/. Acesso em: 7 nov. 2020.

DEVELOPERS, Google. Layouts | Desenvolvedores Android | Android Developers. 2019. **Android Developers**. Available at: https://developer.android.com/guide/topics/ui/declaring-layout. Acesso em: 1 nov. 2020.

EYGM LIMITED. Global telecommunications Scene-setting : the transformation imperative for telcos. n. September, 2019. Available at: https://www.ey.com/Publication/vwLUAssets/ey-accelerating-the-intelligent-enterprise/$FILE/ey-accelerating-the-intelligent-enterprise.pdf. Acesso em: 5 jun. 2020.

FASTLANE. Fastlane - App automation done right. 2020. Available at: https://fastlane.tools/. Acesso em: 29 out. 2020.

FLUTTERANDO. Home - Flutterando. 2019. Available at: https://flutterando.com.br/. Acesso em: 7 nov. 2020.

FORD, Denae; PARNIN, Chris. Exploring causes of frustration for software developers. 23 jul. 2015. **Proceedings - 8th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015** [...]. [*S. l.*]: Institute of Electrical and Electronics Engineers Inc., 23 jul. 2015. p. 115–116. DOI 10.1109/CHASE.2015.19. Available at: http://ieeexplore.ieee.org/document/7166102/. Acesso em: 16 maio 2020.

FREIRE, Alexandre; MOREIRA, André; FERREIRA, Rafael; LESSINGER, Rodrigo; MARACCINI, Victor; ANDRADE, Vinícius; BORGES, Igor; DUBAS, Luiz; MIORIM, Max; RING, Rafael; FIUZA, Rodolfo; MOURA, Thiago; LÚCIO, Wilker Alexandre; FREIRE, Ana Luisa; BAVATI, Ana; VALEIJE, Daniel; DE, Jesus; OLIVEIRA, Eden; FERREIRA, Edward; … FRANÇA, Rafael Maruta. **Mobile Architecture Task Force**. [*S. l.*: *s. n.*], 2019.

GOGGLE. Desenvolvedores Android | Android Developers. 2020. Available at: https://developer.android.com/docs. Acesso em: 4 nov. 2020.

GOOGLE. Arquitetura da plataforma | Desenvolvedores Android. 2020. Available at: https://developer.android.com/guide/platform. Acesso em: 5 jun. 2020.

GOOGLE. Automatizar testes de interface do usuário | Desenvolvedores Android. 2019a. Available at: https://developer.android.com/training/testing/ui-testing?hl=pt-br. Acesso em: 5 nov. 2020.

GOOGLE. **dart programming language**. [*S. l.*: *s. n.*], 2013. Available at: https://dart.dev/. Acesso em: 6 abr. 2020.

GOOGLE. Showcase - Flutter. 2019b. Available at: https://flutter.dev/showcase. Acesso em: 30 out. 2020.

GOOGLE LLC. Layouts in Flutter | Flutter Docs. 2019. Available at: https://flutter.dev/docs/development/ui/layout. Acesso em: 1 nov. 2020.

GRONLI, Tor Morten; HANSEN, Jarle; GHINEA, Gheorghita; YOUNAS, Muhammad. Mobile application platform heterogeneity: Android vs windows phone vs iOS vs Firefox OS. maio 2014. **Proceedings - International Conference on Advanced Information Networking and Applications, AINA** [...]. [*S. l.*]: IEEE, maio 2014. p. 635–641. DOI 10.1109/AINA.2014.78. Available at:

http://ieeexplore.ieee.org/document/6838724/. Acesso em: 5 jun. 2020.

HUAWEI. HUAWEI AppGallery-App Creation and Management Guide. 2020. Available at: https://developer.huawei.com/consumer/en/doc/distribution/app/30204. Acesso em: 28 out. 2020.

HUDLI, Arvind; HUDLI, Shrinidhi; HUDLI, Raghu. An evaluation framework for selection of mobile app development platform. 2015. **MobileDeLi 2015 - Proceedings of the 3rd International Workshop on Mobile Development Lifecycle** [...]. [*S. l.*: *s. n.*], 2015. p. 13–16. DOI 10.1145/2846661.2846678. Available at: http://dx.doi.org/10.1145/2846661.2846678. Acesso em: 3 jun. 2020.

JIM, SIMON, Google Inc; LEX, BEREZHNY, Google Inc; WYATT, ARENT, Google Inc; MICHAEL, PERROTTE, Google Inc; GÜNTER, ZÖCHBAUER, Google Inc; RAJU, BITTER, Google Inc; MICHAEL, BECKLER, Google Inc; ALEXANDRE, ARDHUIN, Google Inc; LUKE, FREEMAN, Google Inc; VINCENT, LE QUÉMÉNER, Google Inc; MIKE, HOOLEHAN, Google Inc; GERMAN, SAPRYKIN, Google Inc; STEFANO, RODRIGUEZ, Google Inc; YUSUKE, KONISHI, Google Inc; FREDRIK, SIMÓN, Google Inc; ALI, BITEK, Google Inc; TETSUHIRO, UEDA, Google Inc; DAN, FIELD, Google Inc; NOAH, GROSS, Google Inc; … FELIX, SCHMIDT, Google Inc. Flutter Documentation - Flutter. 2020. Available at: https://flutter.dev/docs. Acesso em: 6 abr. 2020.

JUNIOR, Vanderlei Freitas; CECI, Flavio; WOSZEZENKI, Cristiane Raquel; GONÇALVES, Alexandre Leopoldo. Design science research methodology enquanto estratégia metodológica para a pesquisa tecnológica. **Espacios**, v. 38, n. 6, 2017. Available at: https://www.revistaespacios.com/a17v38n06/a17v38n06p25.pdf. Acesso em: 31 maio 2020.

KERSHELL. 382px-kernel_layout_svg.png (382×302). 2011. Available at: https://kershell.files.wordpress.com/2011/05/382px-kernel_layout_svg.png. Acesso em: 6 jun. 2020.

KITCHENHAM, B.; KITCHENHAM, B.; CHARTERS, S. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2007. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471. Acesso em: 3 jun. 2020.

KITCHENHAM, Barbara; PEARL BRERETON, O.; BUDGEN, David; TURNER, Mark; BAILEY, John; LINKMAN, Stephen. Systematic literature reviews in software engineering - A systematic literature review. **Information and Software Technology**,

v. 51, n. 1, p. 7–15, 1 jan. 2009. https://doi.org/10.1016/j.infsof.2008.09.009.

KLUBNIKIN, Andrei. I Want to Create an App. Where Should I Start? 2019. Available at: https://r-stylelab.com/company/blog/mobile-technologies/i-want-to-create-an-app-where-should-i-start. Acesso em: 6 abr. 2020.

LÖFFLER, Renate; GÜLDALI, Baris; GEISEN, Silke. Towards Model-based Acceptance Testing for Scrum Sprint. **Softwaretechnik-Trends**, v. 30, n. 3, 2010. .

MAINKAR, Prajyot; GIORDANO, Salvatore. **Google Flutter Mobile Development Quick Start Guide: Get up and running with iOS and Android mobile app development**. [*S. l.*: *s. n.*], 2019.

MASCETTI, Sergio; DUCCI, Mattia; CANTÙ, Niccoló; PECIS, Paolo; AHMETOVIC, Dragan. Developing Accessible Mobile Applications with Cross-Platform Development Frameworks. 14 maio 2020. Available at: http://arxiv.org/abs/2005.06875. Acesso em: 29 maio 2020.

MATYUNINA, Julia. Step-by-Step Guide to Testing Flutter Apps | Mobindustry. 2020. Available at: https://www.mobindustry.net/flutter-automated-testing-step-by-step-guide/#close. Acesso em: 30 out. 2020.

NANJAPPAN, V; LIANG, HN; LAU, K; … J Choi - 2017 International; 2017, Undefined. **Clothing-based Weareble Sensors for Unobtrusive Integraction with Mobiel Devices**. [*S. l.*: *s. n.*], 2017. Available at: https://ieeexplore.ieee.org/abstract/document/8368837/. Acesso em: 6 abr. 2020.

NASEHI, Seyed Mehdi; SILLITO, Jonathan; MAURER, Frank; BURNS, Chris. What makes a good code example?: A study of programming Q&A in StackOverflow. set. 2012. **IEEE International Conference on Software Maintenance, ICSM** [...]. [*S. l.*]: IEEE, set. 2012. p. 25–34. DOI 10.1109/ICSM.2012.6405249. Available at: http://ieeexplore.ieee.org/document/6405249/. Acesso em: 8 nov. 2020.

NUNKESSER, Robin. Beyond web/native/hybrid: A new taxonomy for mobile app development. 27 maio 2018. **Proceedings - International Conference on Software Engineering** [...]. [*S. l.*]: IEEE Computer Society, 27 maio 2018. p. 214–218. https://doi.org/10.1145/3197231.3197260.

ORACLE. Oracle Java SE Support Roadmap. 2020. **Oracle**. Available at: https://www.oracle.com/java/technologies/java-se-support-roadmap.html. Acesso em: 5 nov. 2020.

OZGUL, A; TULJAPURKAR, S; BENTON, T G; PEMBERTON, J; TIM H CLUTTON-BROCK, T H; COULSON, T N. Open Handset Alliance. **Technology**, v.

35, n. 5, p. 1–15, 2010. Available at: http://www.openhandsetalliance.com/. Acesso em: 5 jun. 2020.

PEFFERS, Ken; TUUNANEN, Tuure; ROTHENBERGER, Marcus A; CHATTERJEE, Samir. A design science research methodology for information systems research. **Journal of Management Information Systems**, v. 24, n. 3, p. 45–77, dez. 2007. DOI 10.2753/MIS0742-1222240302. Available at: https://www.tandfonline.com/action/journalInformation?journalCode=mmis20. Acesso em: 31 maio 2020.

QUORA. Axel Rietschin's answer to Which programming language is used for making Windows 10? 2020. Available at: https://www.quora.com/Which-programming-language-is-used-for-making-Windows-10/answer/Axel-Rietschin. Acesso em: 16 maio 2020.

RICK WU. (12) Mobile App Security: Native v.s. Flutter | LinkedIn. 2019. Available at: https://www.linkedin.com/pulse/mobile-app-security-native-vs-flutter-rick-wu/. Acesso em: 30 out. 2020.

RIEGER, Christoph; MAJCHRZAK, Tim A. Towards the definitive evaluation framework for cross-platform app development approaches. **Journal of Systems and Software**, v. 153, p. 175–199, 2019. DOI 10.1016/j.jss.2019.04.001. Available at: https://www.sciencedirect.com/science/article/pii/S0164121219300743. Acesso em: 6 abr. 2020.

ROMAN, Yatsenko. Technical Sciences and It Comparative Analysis of Cross-Platform Frameworks for. **ojs.ukrlogos.in.ua**, , p. 132–136, 2020. Available at: https://ojs.ukrlogos.in.ua/index.php/2617-7064/article/view/223. Acesso em: 29 maio 2020.

SHAH, Kewal; SINHA, Harsh; MISHRA, Payal. Analysis of Cross-Platform Mobile App Development Tools. 1 mar. 2019. **2019 IEEE 5th International Conference for Convergence in Technology, I2CT 2019** [...]. [*S. l.*]: Institute of Electrical and Electronics Engineers Inc., 1 mar. 2019. https://doi.org/10.1109/I2CT45611.2019.9033872.

SR. IOS DEVELOPER. iOS Architecture | IOS Development. 2018. Available at: https://aruniphoneapplication.blogspot.com/2017/01/ios-architecture.html. Acesso em: 5 jun. 2020.

STATCOUNTER. StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share. 2019. Available at: https://gs.statcounter.com/. Acesso

em: 6 abr. 2020.

THE FREEBSD PROJECT. The FreeBSD copyright. 1992. Available at: https://www.freebsd.org/copyright/freebsd-license.html. Acesso em: 20 out. 2020.

THE LINUX INFORMATION PROJECT. Kernel Definition. 2004. **The Linux Information Project**. Available at: http://www.linfo.org/kernel.html. Acesso em: 5 jun. 2020.

TRAN, Thanh. **Flutter Native Performance and Expressive UI/UX**. [*S. l.*: *s. n.*], 2020.

WU, Wenhao. **React Native vs Flutter, cross-platform mobile application frameworks**. 2018. 28 f. Metropolia University of Applied Sciences, 2018. Available at: https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1. Acesso em: 26 maio 2020.

XUJIM. GitHub - alibaba/flutter_boost: FlutterBoost is a Flutter plugin which enables hybrid integration of Flutter for your existing native apps with minimum efforts. 2019. Available at: https://github.com/alibaba/flutter_boost?spm=a2c65.11461447.0.0.2c1839aaw9Kutp. Acesso em: 30 out. 2020.

本文档使用. A Brief Framework Tour - Stateful Widget Lifecycle - 《Flutter by Example》 - 书栈网 · BookStack. 2019. Available at: https://www.bookstack.cn/read/flutterbyexample/aebe8dda4df3319f.md. Acesso em: 30 out. 2020.

APÊNDICE A

Questionário completo de pesquisa com os desenvolvedores:

1)Quantos anos de Experiência com Programação Android/iOS nativo você tem?

a - 1 ano ou menos
b - 2-5
c - Mais que 5

2)Quantos anos de Experiência com Programação em *Flutter* você tem?

a - 1 ano ou menos
b - 2
c - 3 (considerando o alpha)

3)Você tem se sentido confortável usando *Flutter*?

a - Desconfortável
b - Pouco confortável
c - Indiferente
d - Confortável
e - Muito confortável

4)*Flutter* tem uma boa integração com a IDE utilizada?

a - Pior integração em relação ao Nativo.
b - Indiferente em relação ao Nativo.
c - Melhor integração em relação ao Nativo.

5)As mensagens de erro têm ajudado a resolver os problemas?

a - Pouca ajuda em relação ao Nativo.

b - Indiferente em relação ao Nativo.
c - Maior ajuda em relação ao Nativo.

6)Foi necessário adicionar camadas extras de segurança a aplicação por estar usando *Flutter*?

a - Sim
b - Não

7)Houve muita dificuldade inicial para entender a estrutura lógica (loops, variáveis e funcionamento do código)?

a - Dificuldade em *Flutter* e facilidade Nativo.
b - Dificuldade em Nativo e facilidade *Flutter*.
c - Facilidade em Nativo e facilidade *Flutter*.

8)Você enfrentou algum problema em *Flutter* que não foi possível resolver online?

a - Sim
b - Não

9)Como você se sente em relação a curva de aprendizado em *Flutter*?

a - Tive problemas constantes e difíceis de resolver.
b - Tive problemas constantes e fáceis de resolver.
c - Tive poucos problemas e difíceis de resolver.
d - Tive poucos problemas e fáceis de resolver.
e - Não tive muitos problemas.

10)A Ui possui aspecto similar ao nativo, se não tem, é possível imita-lo?

a - Possui aspecto similar.
b - Não possui aspecto similar e NÃO é possível imita-lo.
c - Não possui aspecto similar, mas é possível imita-lo.

11)A interação com aplicações feitas em *Flutter* como barras de navegação, rolagem, e gestos possui aspecto similar ao nativo, se não tem, é possível imita-lo?

a - Possui aspecto similar.
b - Não possui aspecto similar e NÃO é possível imita-lo.
c - Não possui aspecto similar, mas é possível imita-lo.

12)Como tem sido a experiência em *Flutter* em relação a aplicações nativas no tempo de carregamento inicial da aplicação

a - Pior que o Nativo
b - Igual ao Nativo
c - Melhor que o Nativo

13)Como tem sido a experiência em *Flutter* em relação a aplicações nativas na velocidade da aplicação para alterar as views?

a - Pior que o Nativo
b - Igual ao Nativo
c - Melhor que o Nativo

14)Como tem sido a experiência em *Flutter* em relação a aplicações nativas nos cálculos resultantes da interação do utilizador

a - Pior que o Nativo
b - Igual ao Nativo
c - Melhor que o Nativo

15)Como tem sido a experiência em *Flutter* em relação a aplicações nativas na percepção da velocidade de acesso à rede

a - Pior que o Nativo
b - Igual ao Nativo

c - Melhor que o Nativo

16)Como tem sido a experiência em *Flutter* em relação a aplicações nativas na percepção de estabilidade

a - Pior que o Nativo

b - Igual ao Nativo

c - Melhor que o Nativo