UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ANÁLISE COMPARATIVA ENTRE OS PADRÕES MVP E MVVM NA PLATAFORMA ANDROID

ALEXANDRE FREITAS CESAR

JOÃO PESSOA

2019

UNIVERSIDADE FEDERAL DA PARAÍBA

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ANÁLISE COMPARATIVA ENTRE OS PADRÕES MVP E MVVM NA PLATAFORMA ANDROID

ALEXANDRE FREITAS CESAR

JOÃO PESSOA

CENTRO DE INFORMÁTICA

2019

Alexandre Freitas Cesar

UMA ANÁLISE COMPARATIVA ENTRE OS PADRÕES MVP E MVVM NA PLATAFORMA ANDROID

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em ciência da computação.

Orientador: Raoni Kulesza

Setembro de 2019

Ficha catalográfica: elaborada pela biblioteca do CI.

Será impressa no verso da folha de rosto e não deverá ser contada.
Se não houver biblioteca, deixar em branco.

UMA ANÁLISE COMPARATIVA ENTRE OS PADRÕES MVP E MVVM NA PLATAFORMA ANDROID

ALEXANDRE FREITAS CESAR

Trabalho de Conclusão de Curso de Ciência da Computação intitulado Uma análise comparativa entre os padrões MVP e MVVM na plataforma Android de autoria de Alexandre Freitas Cesar, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Raoni Kulesza

Prof.

Prof.

Coordenador(a) do Departamento CI/UFPB

João Pessoa, 21 de setembro de 2019

Centro de Informática, Universidade Federal da Paraíba

Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600

Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

AGRADECIMENTOS

Agradeço ao professor Raoni Kulesza por ter me ensinado os conhecimentos do mercado de trabalho, assim como pelo seu apoio e suporte na pesquisa do meu trabalho de conclusão.

Agradeço ao meu amigo Amaro por me acompanhar em momentos difíceis do curso.

Agradeço às professoras Daniela Coelho, Thais Gaudencio e Danielle Rousy pelo conhecimento e suas personalidades inspiradoras.

Agradeço aos professores Clauirton de Albuquerque, Ronei marcos, Christian Azambuja, José Antônio Gomes pelo conhecimento e suas personalidades fortes e inspiradoras.

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original".

Albert Einstein

RESUMO

O desenvolvimento de aplicações Android tem sido de interesse para pesquisas desde o lançamento do primeiro aparelho Android. Com a evolução das aplicações, o aspecto de desenvolvimento foi impactado profundamente nesse ramo de software, encaminhando requisitos cada vez mais exigentes para atender as necessidades dos usuários. O padrão arquitetural chamado *Model-View-Presenter* (MVP) é o mais popular para aplicações Android, mas não existe informações que mostrem que esse padrão arquitetural apresente um melhor desempenho entre as arquiteturas existentes com a Application Programming Interface (API) atual. O *Architecture Tradeoff Analysis Method* (ATAM) foi utilizado nesta pesquisa para ajudar a escolher uma arquitetura adequada para o desenvolvimento de software. O objetivo da pesquisa é comparar o MVP e o *Model-View-ViewModel* (MVVM) com foco em desempenho. Os resultados mostraram que o MVVM apresenta uma melhor qualidade de desempenho para CPU e memória quando comparado com o MVP.

Palavras-chave: Android, MVP, MVVM, dispositivos móveis, desenvolvimento móvel, arquitetura de software.

ABSTRACT

Android application development has been of interest to research since the launch of the first Android device. With the evolution of applications, the development aspect was deeply impacted in this software branch, leading increasingly demanding requirements to meet the needs of users. The architectural pattern called Model-View-Presenter (MVP) is the most popular for Android applications, but there is no information to show that this architectural pattern performs better between existing architectures with the current Application Programming Interface (API). The Architecture Tradeoff Analysis Method (ATAM) was used in this research to help choose an appropriate architecture for software development. The goal of the research is to compare MVP and Model-View-ViewModel (MVVM) with a focus on performance. The results showed that MVVM has a better performance quality for CPU and memory when compared to MVP.

Key-words: Android, MVP, MVVM, mobile devices, mobile development, software architecture.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	*Associação Brasileira de Normas Técnicas*
GUI	*Graphical User Interface*
MVC	*MODEL-VIEW-CONTROLLER*
MVP	*MODEL-VIEW-PRESENTER*
MVVM	*MODEL-VIEW-VIEW-MODEL*
ATAM	*Architecture tradeoff analysis method*
MB	*MEGABYTE*
HP	*Hewlett-Packard*
IBM	*International Business Machines Corporation*
UFPB	*Universidade Federal da Paraíba*
GB	*GIGABYTE*

SUMÁRIO

1 INTRODUÇÃO

O Android foi apresentado ao mundo em 2005, e ao longo desses anos de existência, a plataforma alcançou sucesso, tornando-se o sistema operacional móvel mais instalado atualmente[1]. No decorrer da evolução do desenvolvimento Android para aplicações móveis, o Google não forneceu um padrão de arquitetura de software para aplicar no desenvolvimento de aplicações[2]. Nesse quesito, a plataforma oferece aos desenvolvedores uma liberdade muito grande para desenvolver aplicações, no entanto essa liberdade pode gerar problemas para testar, desempenho inferior e aumento de código[2]. O principal objetivo deste trabalho é avaliar dois principais padrões arquiteturais para aplicações Android: MVP e MVVM. Para tanto, foi adotado o método ATAM (Architecture tradeoff analysis method) e ferramentas de análise de execução de aplicações para analisar comparativamente a qualidade do código e os detalhes na estrutura de implementações do uso desses dois padrões arquiteturais.

1.1 Definição do Problema

As aplicações atuais são construídas usando arquiteturas específicas, no qual diferentes arquiteturas oferecem diversas vantagens que vêm com distintas incapacidades[3]. A aplicação de uma arquitetura depende muitas vezes dos requisitos específicos do projeto e propriedades do seu Framework[3]. Os padrões de arquitetura geralmente têm propriedades distintas para atender diferentes tipos de necessidade, algumas arquiteturas mostram melhor desempenho dependendo da sua atividade, o que pode deixar as funcionalidades mais fluidas[3].

Os padrões arquiteturais são um campo de conhecimento que foi desenvolvido durante os anos para melhorar a qualidade dos sistemas[4]. Este conhecimento vai ser importante no futuro, pois novos sistemas podem exigir um padrão arquitetural com qualidade de desempenho. A arquitetura quase sempre é representada por um conjunto de propriedades estáticas das camadas e uma comparação pode ser feita em cima dessas, no entanto não basta apenas comparar suas propriedades, é necessário analisar os resultados de desempenho, no qual esta análise pode ser usada para tomar decisões sobre qual arquitetura usar em um projeto[4]. Para tomar decisões é necessário escolher um método de análise que ajude a escolher um padrão arquitetural adequado para desempenho. Os métodos ATAM e *Software Architecture Analysis Method* (SAAM) podem ajudar a escolher uma arquitetura adequada e consiste em reunir as partes interessadas para analisar[4].

1.2 Objetivo geral

A intenção deste trabalho é comparar os principais padrões arquiteturais usados pelos desenvolvedores para construir aplicações na plataforma Android. O presente trabalho tem como principais objetivos específicos:

1.2.1 Objetivo específico

- Utilizar o método *Architecture tradeoff analysis method* (ATAM) para comparar arquiteturas.
- Conceituar e identificar os padrões arquiteturais mais conhecidos segundo autores.
- Diferenças e semelhanças entre *Model-View-Presenter*(MVP) e *Model-View-ViewModel*(MVVM).
- Avaliar e comparar o desempenho do MVP e MVVM na plataforma Android.

1.3 Motivação

O Android é o sistema operacional móvel mais utilizado do mundo, em uma pesquisa feita em 2013 foi mostrado que 71% dos programadores para aplicações móveis desenvolviam para Android[1]. Deste modo, uma investigação nesta área de desenvolvimento de software será favorável para os desenvolvedores de aplicações móveis. É possível notar que novas aplicações Android estão constatemente sendo lançadas no mercado com mais desempenho[4]. A arquitetura mais tradicional utilizada em aplicações Android é o MVP, mas existe outras arquiteturas como o MVVM[4].

Uma motivação importante para esta pesquisa é que os dados sobre o desempenho das arquiteturas MVP e MVVM não foram feitos com a API atual e ferramentas que mostrem o desempenho em tempo real. Uma análise de desempenho com está ferramenta pode mostrar resultados favoráveis para os desenvolvedores de aplicações móveis e outros pesquisadores. A pesquisa tem como objetivo descobrir se existe um desempenho melhor do MVVM quando comparado com o MVP.

1.4 Grupos Alvo

Esta seção consiste de apresentar quem são os stakeholders da pesquisa. O grupo alvo são os desenvolvedores de aplicações Android que buscam um melhor desempenho da sua aplicação. Conhecendo os padrões arquiteturais baseado nas métricas importantes, os desenvolvedores poderão ter o conhecimento sobre qual arquitetura deveria escolher para

solucionar problemas de desempenho presentes na sua aplicação. A decisão da escolha da arquitetura poderá afetar o desempenho da aplicação desenvolvida. Os pequisadores também são um grupo alvo desta pesquisa.

1.5 Organização do trabalho

Este trabalho está estruturado em 8 capítulos. Este atual descreve a introdução, categorizado Capítulo 1. O Capítulo 2 descreve a identificação dos padrões arquitetuturais existentes e conceitos importantes para entender o funcionamento do Android. O Capítulo 3 descreve o método utilizado na pesquisa para fazer a análise dos padrões arquiteturais. O Capítulo 4 descreve os cenários de uso utilizados na pesquisa para escolher as ações que são necessárias para testes. O Capítulo 5 descreve as aplicações utilizadas para testes de desempenho de CPU, memória e casos de teste. O Capítulo 6 descreve a apresentação dos resultados. O Capítulo 7 descreve o relatório dos resultados. O Capítulo 8 descreve qual o padrão se apresentou melhor diante dos cenários analisados em tempo de execução e memória.

2 CONCEITOS GERAIS

Neste capítulo é descrito os principais conceitos do Android, no qual os padrões arquiteturais tradicionais são identificados, conceituados e diferenciados.

2.1 Constituição de uma aplicação Android

O sistema operacional Android é contruído sobre alguns dos principais componentes e camadas. Os componentes têm o objetivo de ajudar a desenvolver aplicações robustas, testavéis e sustentáveis[5]. O Google publicou esses componentes com o intuito de auxiliar no desenvolvimento de aplicativos para o Android. Nesta seção será descrito três conceitos importantes neste trabalho: *Activity, Fragment* e Recursos.

2.1.1 Activity

É a maneira gráfica de interagir com o usuário, ou seja, uma *Activity* é basicamente uma interface de uma aplicação, para cada nova tela que é criada é necessário uma *Activity*[6]. As *Activitys* quando são abertas utilizam o conceito de pilha, exemplo: quando se tem uma tela e é acessado uma nova tela, a tela nova fica sobre a tela antiga[6]. Dentro da *Activity* é carregado um arquivo xml que compõe a interface.

2.1.1.1 Ciclo de vida

A maioria dos componentes de apps Android têm ciclos de vida ligados a eles, que são gerenciados diretamente pelo próprio sistema.

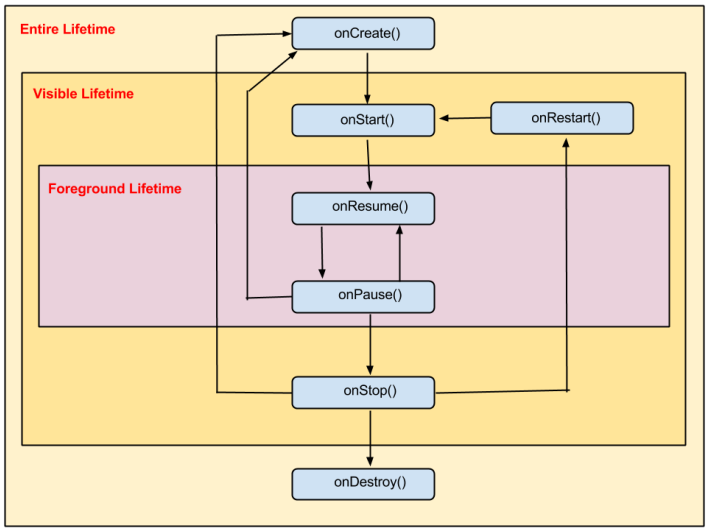

Figura 1 – Ciclo de vida de uma Activity

O ciclo de vida é dividido em 3 níveis: *Entire lifetime, Visible lifetime e Foreground lifetime*[7]. O *Entire lifetime* aborda todo o ciclo do *created* até o *destroyed*. O *Visible lifetime* aborda só a parte que o usuário pode ver a sua atividade que é do *start* até o *stop*. O *Foreground Lifetime* aborda todo contexto que permite interagir com a atividade.

Neste sentido quando o usuário executa a aplicação, a atividade começa no *ONCREATE* que é chamado apenas uma única vez, o usuário ainda não pode ver nada na tela, logo depois a atividade vai para o *ONSTART* no qual inicia a apresentação da tela, mas não pode interagir com a tela, no *ONRESUME* o usuário já está em execução e é possível interagir[7].

2.1.2 Fragment

É uma mini *Activity* que possui o seu próprio ciclo de vida e estão interligados a uma activity, mas não é um componente do sistema, pois não tem assinatura no AndroidManifest[8]. O funcionamento é parecido como as *Activity's* funcionam. A diferença entre ambos é que a Activity é um componente do Android e tem uma conexão com o processo da aplicação, já o *Fragment* fica sobre o encargo direto da *Activity*[8].

2.1.3 Recursos

Os recursos são os arquivos adicionais e conteúdo estático que o código usa. Existe vários tipos de recursos como: *layout, bitmaps*, dimensão e etc.

2.2 Padrões de arquitetura

No decorrer de toda a história do desenvolvimento Android, o Google não entregou sua posição sobre qual o padrão de arquitetura de software que deveria utilizar para desenvolver aplicativos Android[2]. Quando lançaram oficialmente as primeiras versões do Android, o objetivo era usar o MVC para o desenvolvimento das aplicações, mas rapidamente surgiram problemas nesse padrão arquitetural, então foi preciso criar novas abordagens como os padrões MVP e MVVM[2].

2.2.1 Arquitetura multicamadas

O sistema multicamadas não tem uma definição clara de quais são as camadas que certamente devem ter em um aplicativo Android, nem mesmo o número mínimo e máximo. Este sistema faz uso de objetos distribuídos (permite a operação com objetos remotos) associado à utilização de interfaces para executar seus procedimentos, podendo esta aplicação Android ser dividida em várias partes, cada uma bem definida, com suas características e responsável por determinadas funções[9]. Em um aplicativo nesta forma,

pelo menos três camadas são necessárias: apresentação, regras de negócios e banco de dados.

2.2.2 Model-View-Controller - MVC

O MVC foi criado na década de 70 como forma de separar a lógica de apresentação da lógica de negócios, reduzindo a ligação entre classes. No ano de 2009, quando saiu as primeiras versões do Android, a ideia era usar o padrão arquitetural MVC para o desenvolvimento dos projetos[2]. O incentivo para usar o MVC foi os desenvolvedores conseguirem paralelizar o desenvolvimento e conseguir fazer o que chamamos de reuso de código[10]. Esta abordagem separa sua aplicação em um nível de composição com 3 conjuntos de funções, conforme Figura 2.

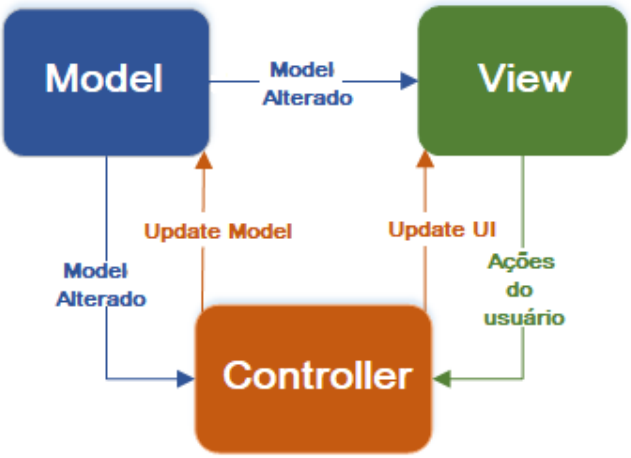

Figura 2 – Arquitetura MVC

A Camada *View* é responsável por apresentar informações de forma gráfica para o usuário e faz a comunicação com o controller. A Camada *Controller* é responsável por fazer a ligação entre a interface gráfica e a lógica de negócios, alterando atributos e mostrando as modificações na interface gráfica. A Camada *Model* é onde a lógica de negócio é definida, incluindo a persistência de dados[10].

O MVC consegue separar o *Model* e a *View*. Deste jeito o *Model* pode ser testado, pois não está vinculado a nada e a View não tem muito para testar em um nível de teste unitário. No entanto, o *Controller* não vai ficar disponível para testes, pois várias partes que deveriam estar dentro do *Controller* foram passadas para o *Activity*[10].

2.2.3 Model-View-Presenter - MVP

O MVP surgiu na década de 90, no qual a união das empresas multinacionais APPLE, HP e IBM fez com que esse padrão arquitetural surgisse para complementar

necessidades que o padrão MVC não cobria[11].

O MVP possibilita uma divisão melhor para camadas quando o objetivo é separar camadas superiores de camadas inferiores, ou seja, possibilita que a camada diretamente relacionada com a interface do usuário somente se comunique com a camada diretamente abaixo dela[10]. Esta abordagem separa sua aplicação em um nível de composição com 3 conjuntos de funções, conforme Figura 3.

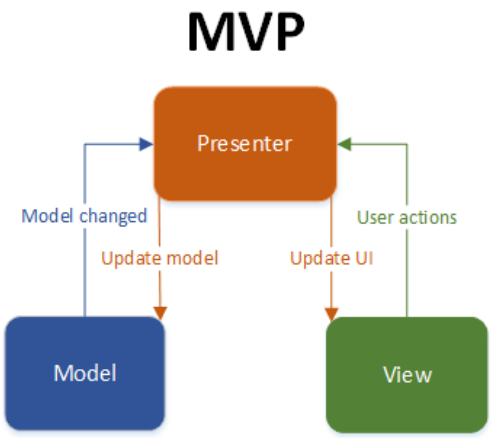

Figura 3 – Arquitetura MVP

A camada *View* é responsável por responder a saída ou entrada de dados, no qual a saída vem da camada Presenter, e a entrada vem normalmente do usuário[10]. A camada *Presenter* é responsável por responder as invocações da camada de visualização e invocações da camada de modelo e pode também invocar ambas as camadas[10]. O *Presenter* pode incluir lógica de negócio e pode ser responsável pela formatação dos dados[2]. A camada *Model*: é responsável por fornecer os dados e pode conter lógica de negócio para domínio do problema[10].

A grande vantagem do padrão arquitetural MVP é que ele descreve três níveis de abstração, o que torna mais fácil para depurar a aplicação Android[10]. A separação da lógica de negócios e lógica de acesso a dados também permite o teste que vai ser implementado. A camada *Presenter* faz referência ao *Controller* do MVC, exceto por ele não estar vinculado ao *View*, apenas a uma interface, com isto ele não mais gerencia o tráfego de solicitações recebidas, como é feito no *Controller*[10].

2.2.4 Model-View-ViewModel - MVVM

O MVVM é um padrão que foi criado em 2005, por John Gossman. O MVVM assemelha-se em alguns aspectos do MVC e MVP[12]. Esta abordagem separa sua aplicação em um nível de composição com 3 conjuntos de funções, conforme Figura 4.

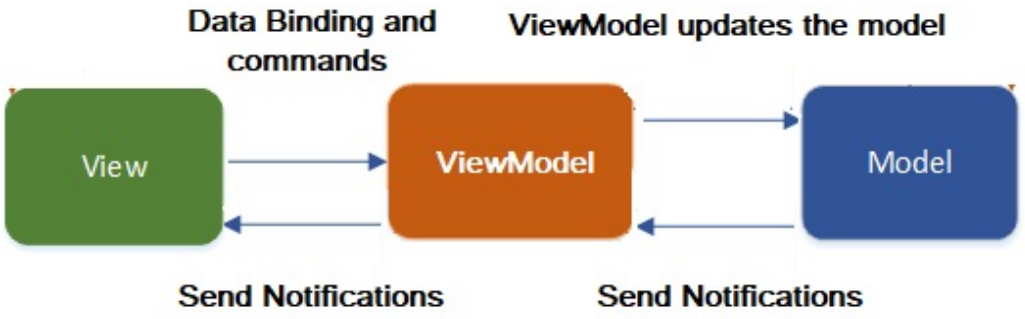

Figura 4 – Arquitetura MVVM

A responsabilidade da *ViewModel* no contexto do MVVM, é disponibilizar para a *View* uma lógica de apresentação[10]. A *ViewModel* não tem nenhum conhecimento específico sobre a *View*, ou como ela implementada, nem o seu tipo[10]. A *ViewModel* implementa propriedades e comandos, para que a View possa preencher seus controles e notifica a mesma, caso haja alteração de estado, seja através de eventos ou notificação de alteração[10]. A *ViewModel* é peça fundamental no MVVM, por que é ela quem vai coordenar as iterações da *View* com o *Model*, considerando que ambos não terem conhecimento um do outro[10]. E além de tudo isto, a *ViewModel*, também pode implementar a lógica de validação, para garantir a consistência dos dados[10]. A camada *Model* é responsável por fornecer os dados e pode conter lógica de negócio para domínio do problema.

2.3 Diferenças e semelhanças entre MVP e MVVM

Os padrões arquiteturais MVP e MVVM apresentam poucas diferenças em estrutura e responsabilidades dos componentes, mas que podem resultar em diferenças de desempenho. Os seus componentes são similares, mas com responsabilidades diferentes.

1. As diferenças entre MVP e MVVM, conforme Churg (2019)[15]:

    - A camada *Presenter* e a camada *View* estão em um relacionamento de 1 para 1(significa dizer que um Presenter pode mapear apenas 1 View).
    - O *View* e o *ViewModel* estão em um relacionamento de 1 para muitos(significa dizer que uma *ViewModel* pode mapear muitas *Views*).
    - A camada *ViewModel* substitui o *Presenter* na camada intermediária.
    - A camada *Presenter* mantém referências à camada *View*, já a camada *ViewModel* não.
    - A camada *ViewModel* envia fluxos de dados.

- O *ViewModel* não sabe que o *View* está escutando.

2. Ainda Churg cita as semelhanças entre MVP e MVVM:

   - A camada *Model* é semelhante para MVP e MVVM.

2.4 Trabalhos relacionados

Esta seção mostra os temas relacionados com esta pesquisa.

O pesquisador Tian Lou fez a pesquisa "*A comparison of Android Native App Architecture MVC, MVP and MVVM*". O pesquisador utilizou as métricas: desempenho, modificabilidade e capacidade de teste. Os dados obtidos foram comparados e analisados para definir a melhor abordagem de padrão arquitetural. A análise de experimentos mostram que MVP e MVVM são melhores que MVC[3]. A pesquisa também mostrou que o MVP tem melhor modificabilidade e o MVVM proporciona uma melhor capacidade de teste[3].

O pesquisador Vladyslav Humeniuk fez a pesquisa "*Android Architecture Comparison: MVP vs. VIPER*". O pesquisador utilizou as métricas: desempenho, modificabilidade e capacidade de teste. Os dados obtidos foram comparados e analisados para definir a melhor abordagem de padrão arquitetural. A análise de experimentos mostrou que VIPER apresenta um melhor desempenho que o MVP e o MVP mostrou um código mais simples de entender com menos linhas de código[4].

3 METODOLOGIA

O trabalho apresenta uma metodologia descritiva, no qual o objetivo é esclarecer ao máximo um assunto já conhecido. Neste caso foi feito uma forte revisão teórica envolvendo o objeto de estudo que é comparar e analisar as informações do desempenho de CPU e memória.

O método ATAM foi adotado nesta pesquisa afim de analisar comparativamente a qualidade do código e os detalhes na estrutura de implementações da mesma aplicação utilizando padrões arquiteturais diferentes. O método ATAM analisa quão bem a arquitetura de software satifaz objetivos particulares de qualidade. Este método é baseado no *Software Architecture Analysis Method* (SAAM) e é considerado uma evolução deste método. Foi utilizado uma versão adaptada para pesquisa feita pelo pesquisador Lou[3]. O método é formado por 4 fases e 9 passos, sendo eles:

3.1 Fase de apresentação

- Apresentação do ATAM para os stakeholders. Neste passo é escolhido o grupo alvo para pesquisa.
- Descrever os objetivos de négocio que estão motivando o desenvolvimento e quais serão os requisitos arquiteturalmente relevantes. Neste passo é mostrado a motivação para pesquisa.
- Apresentar a arquitetura proposta. Neste passo é mostrado o funcionamento das camadas do MVVM.

3.2 Fase de investigação e análise

- Identificar e descrever as abordagens arquiteturais tradicionais.
- Coletar cenários que são populares e identificar as principais funções básicas. Neste passo foram coletados alguns cenários que são populares na *Play Store* para encontrar ações básicas.
- Coletar os requisitos. Neste passo o requisito utilizado é o desempenho.

3.3 Fase de teste

- Brainstorm e priorização de cenários. Neste passo a aplicação TODO e Finder foram escolhidas para análise.

- Descrição dos experimentos. Neste passo as aplicações TODO e finder foram testadas para análise.

3.4 Fase de relátorio

- Apresentar resultados. Neste passo é mostrado um relátorio com base nas informações recolhidas da descrição do experimento.

4 CENÁRIOS DE USO

Um passo necessário do método ATAM é o levantamento dos cenários de uso, ele é um conjunto de ações básicas que é necessário para completar um caso de uso, são as ações são descritas do ponto de vista da aplicação[4]. As ações básicas que vão ser realizadas durantes os testes são as seguintes: clicar no botão, acessar armazenamento de dados e memória, entrar em uma nova tela, selecionar os itens da lista e etc. As aplicações mais conhecidas do Android geralmente são formadas por cenários de caso de uso pequenos que consistem em até cinco ações básicas[4].

4.1 Aplicativo Youtube Music

É uma das aplicações criadas pelo Google. Esta aplicação permite que o usuários procurem e escutem músicas. Este cenário de uso é o mais utilizado no Youtube Music[1] . As ações básicas:

- Tela principal aberta
- Carregar músicas recentemente escutadas
- Clicar no ícone "lupa"para pesquisar uma música
- Clicar no ícone da música
- Abrir nova tela para transmitir a música

4.2 Aplicativo Tarefas

É uma das aplicações criadas pelo Google. O aplicativo Tarefas[2] permite que os usuários registrem tarefas. As ações básicas:

- Tela principal aberta
- Clicar no botão "+"para registrar uma tarefa
- Escrever uma nova tarefa
- Clicar no botão para registrar detalhes

[1] Aplicativo Youtube Music. Disponível em: < https://play.google.com/store/apps/details?id=com.google.android.apps.youtube.music>.

[2] Aplicativo Tarefas. Disponível em: <https://play.google.com/store/apps/details?id=com.google.android.apps.tasks>.

- Clicar no botão salvar
- Abrir nova tela para listar tarefas salvas

Outro cenário de caso de uso é editar a sua tarefa. As ações básicas:

- Tela principal aberta
- Clicar na tarefa
- Abrir nova tela para editar tarefa
- Editar nome ou descrição da tarefa
- Confirmar edição
- Abrir nova tela para listar tarefas salvas

Outro cenário de caso de uso é ver o feed do usuário. As ações básicas:

- Tela principal aberta
- Tarefas registradas na tela principal
- Rolar para ver tarefas

Outro cenário de caso de uso é deletar as tarefas. As ações básicas:

- Tela principal aberta
- Clicar na tarefa
- Abrir nova tela com descrição e nome da tarefa
- clicar no ícone da lixeira para deletar
- Voltar para tela principal

4.3 Aplicativo Medium

O aplicativo Medium[3] permite que o usuário pesquise para encontrar artigos explicando vários assuntos. O cenário de uso mais popular é a pesquisa. Este cenário é composto pelas seguintes ações básicas:

[3] Aplicativo Medium. Disponível em: <https://play.google.com/store/apps/details?id=com.medium.reader>.

- Tela principal aberta
- Digitar o texto da pesquisa
- Clicar na lupa para pesquisar
- Abrir nova tela para listar sua pesquisa
- Clicar no ícone da imagem da pesquisa para ver o texto
- Abrir nova tela para exibir a pesquisa

É importante notar que os cenários de uso mais populares consistem de ações básicas, ou seja, a maioria dos cenários de caso de uso têm como finalidade permitir que o usuário alcance os seus objetivos mais rapidamente[4]. Isto significa dizer que os cenários de caso de uso pequenos são de interesse para pesquisadores e desenvolvedores, pois aplicações populares utilizam estes cenários pequenos[4].

As ações básicas mais frequentes encontradas nas aplicações são: criar, deletar, editar, pesquisar e listar. Estas ações também foram notadas pelos pesquisadores Lou e Humeniuk como as mais frequentes das aplicações populares[4].

5 APLICAÇÕES ANDROID PARA O EXPERIMENTO

Neste capítulo é mostrado as aplicações Android que foram utilizadas no experimento. Estas aplicações requerem nível mínimo API Android 14, que corresponde ao Android 4.0.1 - 4.0.2 Ice Cream Sandwich, em torno dos 99% está o mercado que funciona com a versão 4.0 até a 9.0 que é do Android mais atual[4].

5.1 Aplicativo TODO

Este aplicativo foi desenvolvido pelo Google para mostrar diferentes abordagens arquitetônicas para o desenvolvimento de aplicativos no sistema operacional Android. Esta aplicação está disponível nos padrões MVP e MVVM.

A Figura 5 mostra a interface do usuário das telas em caso de uso.

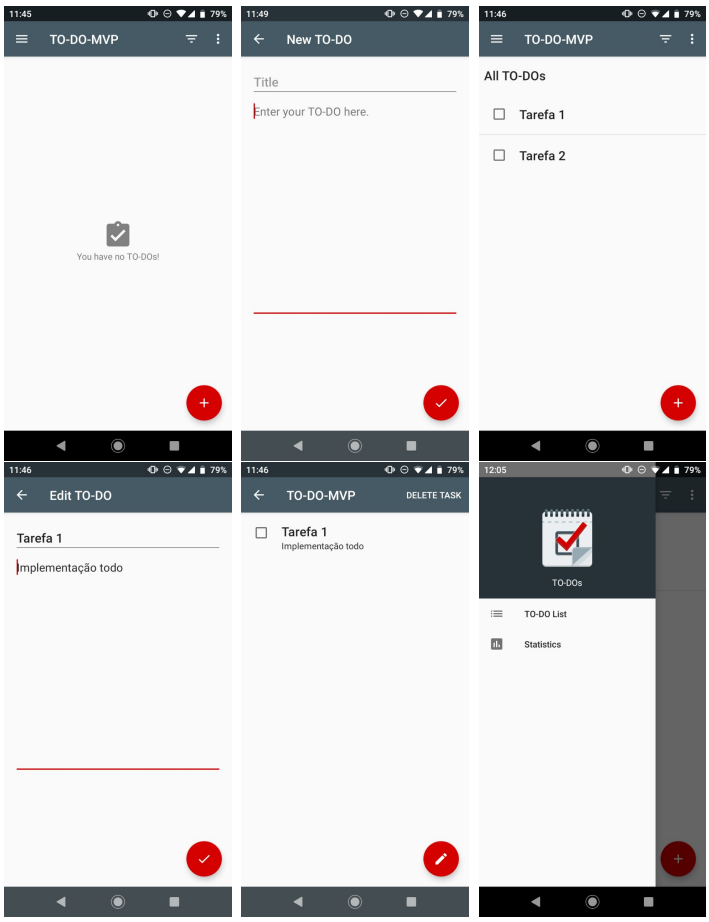

Figura 5 – Aplicativo TODO GOOGLE

A navegação é feita utilizando menu latetal e botões de cadastramento e visualização. O botão vermelho é utilizado para chamar atenção do usuário sobre sua funcionalidade principal que é cadastrar e editar. O menu é utilizado para mostrar estatísticas e voltar para o menu principal com a lista de tarefas.

5.2 Aplicativo Movie Finder

Este aplicativo foi desenvolvido pelo DigiGene para comparar diferentes abordagens arquitetônicas para o desenvolvimento de aplicativos no sistema operacional Android. Esta aplicação está disponível nos padrões MVC, MVP, MVVM e sem padrão.

A Figura 6 mostra a interface do usuário das telas em caso de uso.

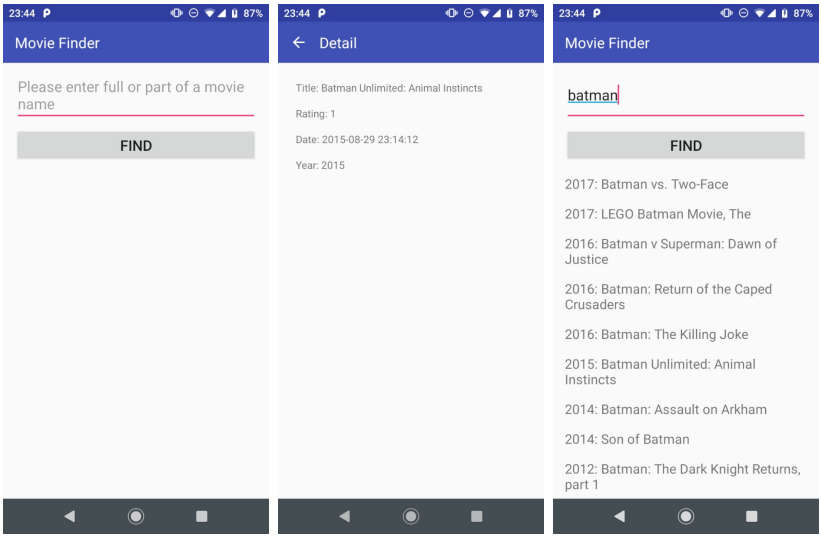

Figura 6 – Aplicativo Movie Finder

A navegação é feita utilizando botões. O botão *finder* é utilizado para encontrar filmes. Clicar no nome do filme que está na lista é mostrado uma descrição.

6 DESCRIÇÃO DO EXPERIMENTO E RESULTADOS DA AVALIAÇÃO

Neste capítulo é mostrado uma descrição com base nas informações recolhidas da fase de investigação. A implementação do relátorio consistiu de um conjunto de casos de uso, que executa operações básicas nas aplicações Android levantadas.

A Tabela 1 mostra as configurações do computador utilizado para executar as ferramentas Profile[1] do Android Studio na versão 3.4.2 e Appium[2] na versão 1.3.2.

Tabela 1 – Computador utilizado para testes

Dell 15r Special Edition 7520	Detalhes
Sistema Operacional	Windows 10
Processador	Intel Core i5 3230M 2,60 GHz (2 núcleos e 4 threads)
Memória	8 GB

A Tabela 2 mostra as configurações do smartphone que executou as aplicações do experimento.

Tabela 2 – Aparelho utilizado para testes

Motorola G(6) play	Detalhes
Sistema Operacional	Android 9.0 (Pie)
Processador	Octa-core 1.8 GHz Cortex-A53
Memória	3 GB
Versão do hardware	P4

Os casos de uso para o experimento serviram para representar os cenários de uso que foram levantados no capítulo 4. Estes cenários têm um conjunto de ações simples como: adicionar, remover, deletar, procurar e listar. As ações que são apenas cálculo de hardware devem ter o acesso rápido, pois não depende do requisito da Internet. Neste capítulo também será mostrado tabelas com medições obtidas nos testes. Os padrões arquiteturais foram avaliados no contexto das medidas e analisadas para responder objetivos da pesquisa e seguir os passos do ATAM. Os padrões de arquiteturas foram avaliados de acordo com a métrica desempenho.

6.1 Inspecionar atividades de CPU para comparar desempenho

O desempenho é uma métrica muito importante para avaliar os padrões arquiteturais[3]. Uma arquitetura com um desempenho comprometido pode levar o usuário a

[1] Ferramenta Profile. Disponível em: <https://androidstudio.googleblog.com/2019/07/android-studio-342-available.html>.

[2] Ferramenta Appium. Disponível em: <https://github.com/appium/appium-desktop/releases/tag/v1.3.2>.

ter um experiência de uso ruim devido aos problemas de congelamento e lentidão quando o aplicativo abre novas telas, então o desempenho deve ser considerado uma métrica principal para uma análise de arquitetura[3].

Profile é uma ferramenta utilizada para medir desempenho de CPU e uso de memória. A ferramenta é capaz de otimizar e mostrar problemas de desempenho presentes, animações que nunca terminanam, problemas de congelamento e alto consumo de energia[16]. Uma otimização do uso da CPU em um aplicativo Android pode deixar a aplicação mais rápida e suave[16]. O uso da ferramenta Profile do Android Studio serviram para inspecionar o uso de CPU das atividades em tempo real, com está ferramenta o tempo de execução das principais funções do aplicativo foram capturadas em microsegundos e estes dados foram utilizados para comparação dos padrões arquiteturais MVP e MVVM.

Os resultados de desempenho foram coletados por meio de uma estrutura de testes automatizados para aplicações Android. Os cliques e digitações foram realizados em uma velocidade constante pela ferramenta Appium[17], 45 medições foram feitas para calcular a média e mediana, a mediana foi realizada para evitar que valores individuais pudessem influenciar nos resultados. O intervalo de confiança foi calculado para prever o intervalo das amostras de cada ação básica. O nível de confiança utilizado foi de 95%, pois este valor é o mais padrão para prever intervalos sem grandes problemas. Os resultados, amostras, tutorial e código estão disponíveis na referência [18].

6.1.1 Descrever o experimento com aplicação TODO da Google

O experimento tem como objetivo comparar arquiteturas pelo desempenho dos casos de uso construídos com padrões arquiteturais que foram levantados. O desempenho é medido em tempo real de execução e memória total utilizada.

A aplicação TODO[3] foi utilizada para avaliar os padrões arquiteturais MVP e MVVM. As avaliações vão ser em cima das funções, conforme a Figura 7.

6.1.1.1 Criar Tarefa

Esta seção mostra o desempenho do uso de CPU e consumo de memória em tempo real para adicionar as tarefas.

A Tabela 3 mostra a média, mediana e desvio da execução em tempo real da função criar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

[3] Aplicação TODO. Disponível em: <https://github.com/googlesamples/android-architecture>.

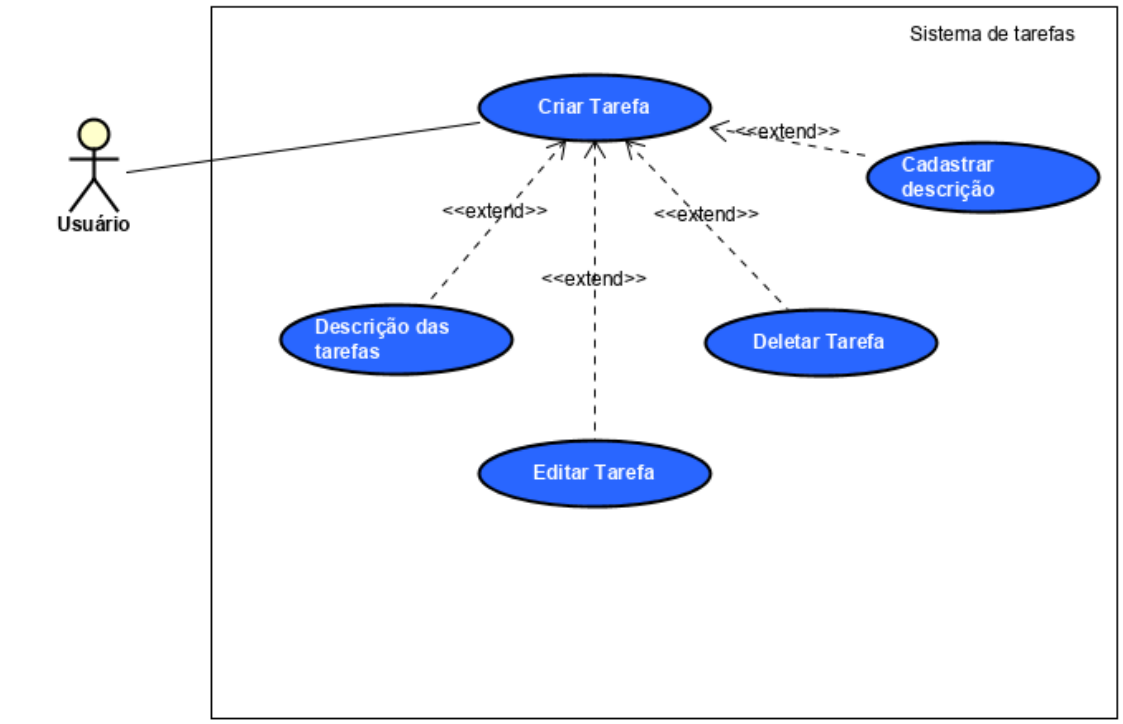

Figura 7 – Diagrama de caso de uso do TODO

Tabela 3 – Resultado do uso de CPU para adicionar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	14.246	14.269	14.137	14.045	5,415
MVVM	14.504	12.107	13.433	12.271	6,502

Observando a média presente na Tabela 3, o padrão arquitetural MVVM mostrou uma redução de 15,15% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 12,63% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

A Figura 8 mostra o desempenho de CPU para criação da tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta Profile do Android Studio.

O intervalo de 95% de confiança para o desempenho de CPU em criar tarefa no MVP varia entre [12.681; 15.854]. O intervalo de 95% de confiança para o desempenho de CPU em criar tarefa no MVVM varia entre [10.202; 14.008].

A Figura 9 mostra o desempenho da memória para criação da tarefa em tempo real para os padrões arquiteturais MVP e MVVM que foi medido utilizando a ferramenta profile do Android Studio.

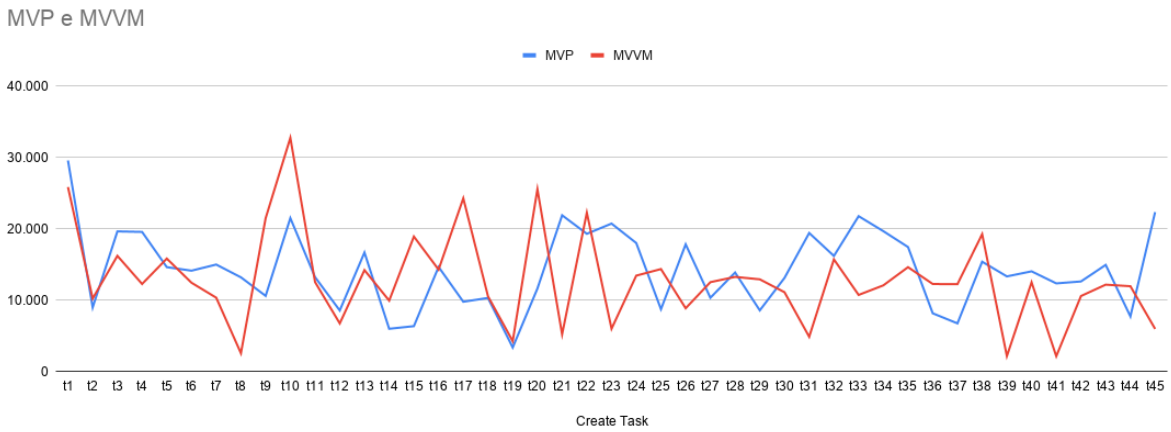

Figura 8 – Desempenho do uso CPU para criar tarefa em microsegundos

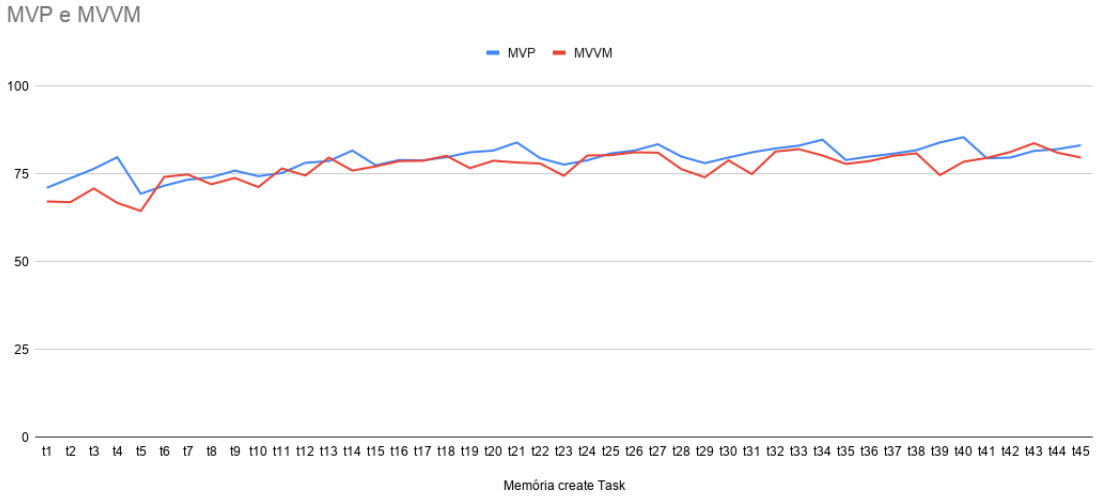

Figura 9 – Desempenho do uso de memória para criar tarefa em MB

O intervalo de 95% de confiança para o desempenho de memória do criar tarefa no MVP varia entre [78.01; 80.18]. O intervalo de 95% de confiança para o desempenho de memória do criar tarefa no MVVM varia entre [75.41; 77.98].

A Tabela 4 mostra a média, mediana e desvio da execução em tempo real da função criar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Tabela 4 – Resultado do uso de memória para adicionar tarefa com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	77.2	79.1	78.1	79.6	3,70
MVVM	74.7	76.7	75.9	78.2	4,41

Observando a média presente na Tabela 4, o padrão arquitetural MVVM mostrou

um redução de 3,03% no tempo real do uso de memória em comparação com o MVP. Observando a mediana, o padrão arquitetural MVVM mostrou uma redução de 1,76% no uso de memória em comparação com o MVP. Do ponto de vista de uso de memória a mudança na quantidade de MB não foi muito significativa, mesmo assim a qualidade do desempenho foi MVP < MVVM, pois o intervalo de confiança mostrou um melhor intervalo para o MVVM.

6.1.1.2 Deletar tarefa

Esta seção mostra o desempenho do uso de CPU e memória em tempo real para deletar as tarefas.

A Tabela 5 mostra a média, mediana e desvio da execução em tempo real da função deletar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Tabela 5 – Resultado do uso de CPU para deletar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	10.363	10.736	11.178	10.302	4,736
MVVM	7.116	6.831	7.317	7.061	1,317

Observando a média presente na Tabela 5, o padrão arquitetural MVVM mostrou uma redução de 36,36% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 31,43% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

A Figura 10 mostra o desempenho de CPU para deletar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta Profile do Android Studio.

O intervalo de 95% de confiança para o desempenho do uso de CPU em deletar tarefa no MVP varia entre [9.359; 12.115]. O intervalo de 95% de confiança para o desempenho do uso de CPU em deletar tarefa no MVVM varia entre [6.442; 7.213].

A Figura 11 mostra o uso de memória para deletar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android Studio.

O intervalo de 95% de confiança no desempenho do uso de memória para deletar tarefa no MVP varia entre [89.4; 96.9]. O intervalo de 95% de confiança no desempenho do uso de memória para deletar tarefa no MVVM varia entre [83.2; 86.9].

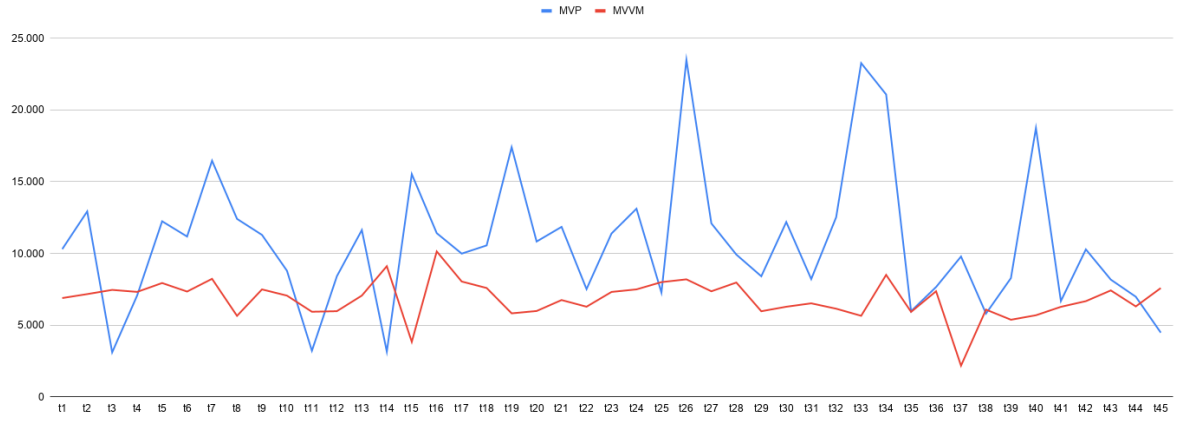

Figura 10 – Desempenho do uso CPU para deletar tarefa em microsegundos

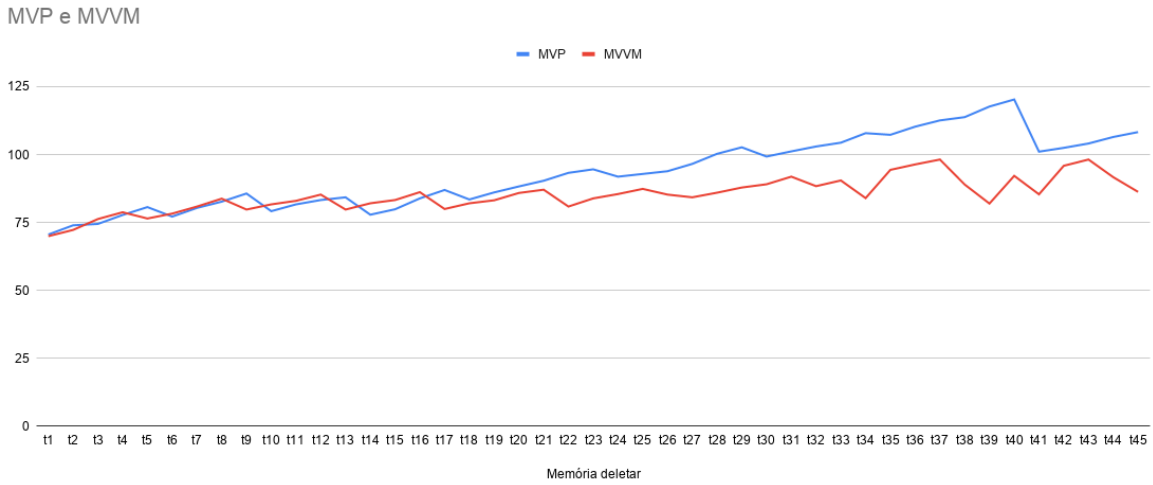

Figura 11 – Desempenho do uso de memória para deletar tarefa em MB

A Tabela 6 mostra a média, mediana e desvio da execução em tempo real da função deletar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM.

Tabela 6 – Resultado do uso de memória para deletar tarefa com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	83.2	93.2	83.3	92.9	12,9
MVVM	81.3	85.1	82.1	85.3	6,23

Observando a média presente na Tabela 6, o padrão arquitetural MVVM mostrou um redução de 8,68% no tempo real do uso de memória em comparação com o MVP. Verificando a mediana, o padrão arquitetural MVVM mostrou uma redução de 8,18% no uso de memória em comparação com o MVP. Do ponto de vista de uso de memória a qualidade do desempenho foi MVP < MVVM.

6.1.1.3 Listar tarefa

Esta seção mostra o desempenho do uso de CPU e memória em tempo real para listar as tarefas.

A Tabela 7 mostra a média, mediana e desvio da execução em tempo real da função listar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Tabela 7 – Resultado do uso de CPU para listar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	10.009	10.302	8.931	9.848	3,583
MVVM	11.172	9.968	9.937	9.158	5,645

Observando a média presente na Tabela 7, o padrão arquitetural MVVM mostrou uma redução de 3,24% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 7,00% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

A Figura 12 mostra o desempenho de CPU para listar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android Studio.

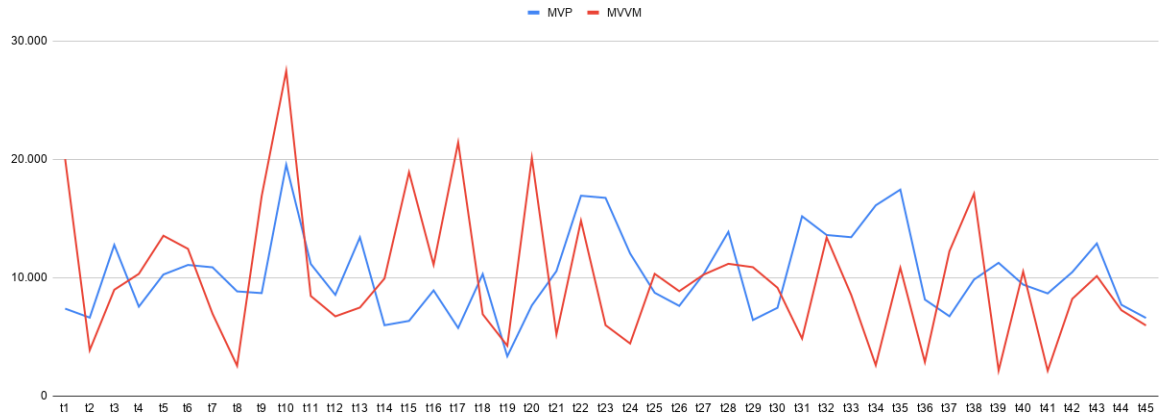

Figura 12 – Desempenho do uso CPU para listar tarefas em microsegundos

O intervalo de 95% de confiança para o desempenho do uso de CPU em listar tarefa no MVP varia entre [9.256; 11.342]. O intervalo de 95% de confiança para o desempenho do uso de CPU em listar tarefa no MVVM varia entre [8.312; 11.613].

6.1.1.4 Editar tarefa

Esta seção mostra o desempenho do uso de CPU em tempo real para editar as tarefas.

A Tabela 8 mostra a média, mediana e desvio da execução em tempo real da função editar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Tabela 8 – Resultado do uso de CPU para editar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	15.830	16.544	15.957	16.713	2,351
MVVM	13.343	13.579	14.525	14.377	4,346

Observando a média presente na Tabela 8, o padrão arquitetural MVVM mostrou uma redução de 17,92% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 13,97% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

A Figura 13 mostra o desempenho de CPU para editar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android Studio.

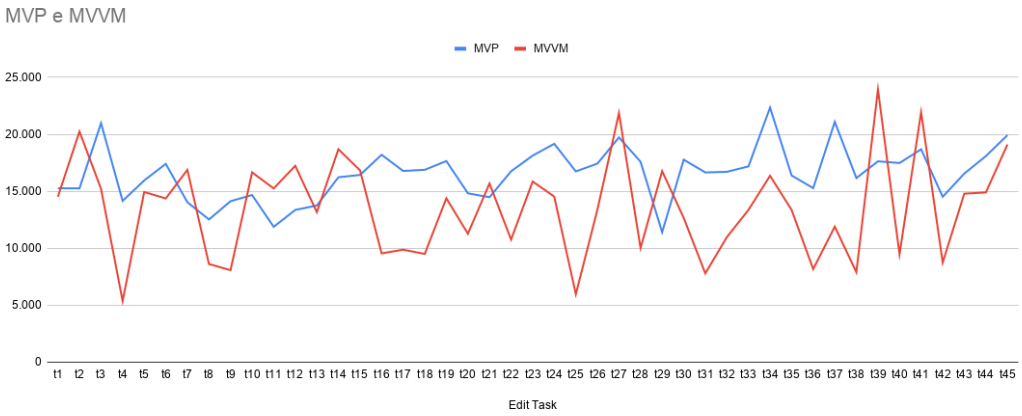

Figura 13 – Desempenho do uso CPU para editar tarefa em microsegundos

O intervalo de 95% de confiança para o desempenho do uso de CPU em editar tarefa no MVP varia entre [15.852; 17.237]. O intervalo de 95% de confiança para o desempenho do uso de CPU em editar tarefa no MVVM varia entre [12.309; 14.848].

A Figura 14 mostra o uso de memória para editar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android Studio.

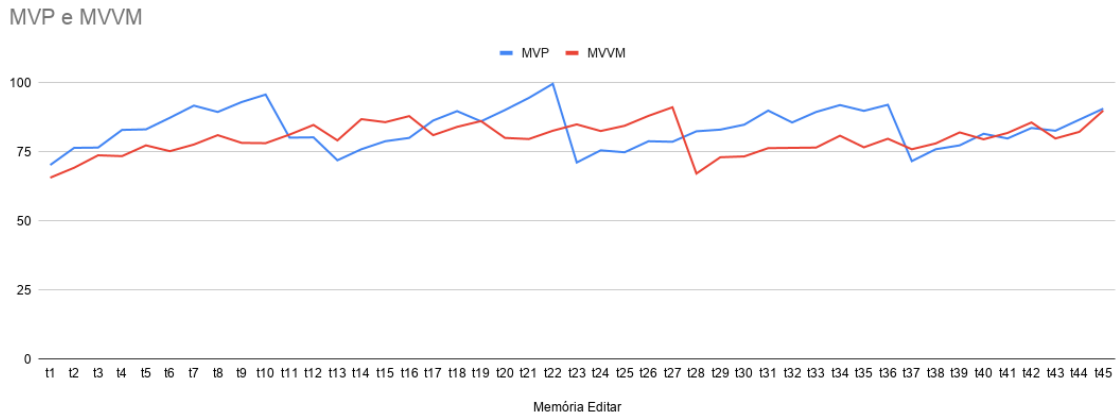

Figura 14 – Desempenho do uso de memória para editar tarefa em MB

O intervalo de 95% de confiança para o desempenho do uso de memória em editar tarefa no MVP varia entre [81.3; 85.6]. O intervalo de 95% de confiança para o desempenho do uso de memória em editar tarefa no MVVM varia entre [78.3; 81.5].

A Tabela 9 mostra a média, mediana e desvio da execução em tempo real da função editar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Tabela 9 – Resultado do uso de memória para editar tarefa com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	83.3	83.5	83.0	83.1	7,19
MVVM	80.0	79.95	81.1	79.9	5,61

Observando a média presente na Tabela 9, o padrão arquitetural MVVM mostrou um redução de 4,25% no tempo real do uso de memória em comparação com o MVP. Verificando a mediana, o padrão arquitetural MVVM mostrou uma redução de 3,85% no uso de memória em comparação com o MVP. Do ponto de vista de uso de memória a mudança na quantidade de MB não foi muito significativa, mesmo assim a qualidade do desempenho foi MVP $<$ MVVM, pois o intervalo de confiança mostrou um melhor intervalo para o MVVM.

6.1.2 Descrever o experimento com aplicação Movie Finder

O aplicativo Movie Finder vai ser utilizado para avaliar os padrões arquiteturais MVP e MVVM. As avaliações vão ser em cima das função pesquisar, conforme a Figura 15.

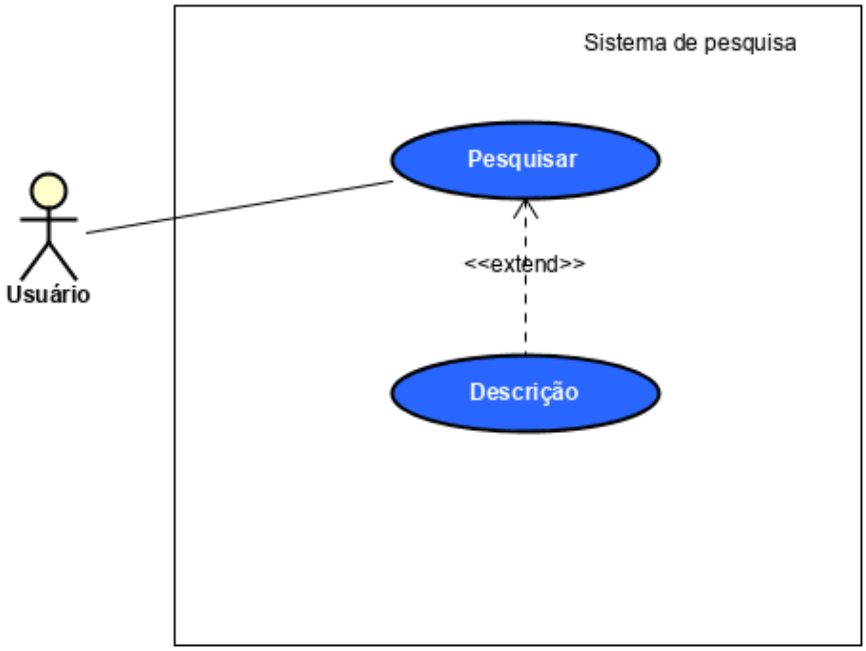

Figura 15 – Diagrama de caso de uso do Movie Finder

6.1.2.1 Pesquisar filmes

Esta seção mostra o desempenho do uso de CPU e memória em tempo real para pesquisar os filmes.

A Tabela 10 mostra a média, mediana e desvio da execução em tempo real da função editar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Tabela 10 – Resultado do uso de CPU para pesquisar filmes com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	9.751	8.768	9.485	7.707	4,03
MVVM	6.599	6.128	5.919	5.728	2,73

Observando a média presente na Tabela 10, o padrão arquitetural MVVM mostrou uma redução de 30,10% no tempo real de CPU em comparação com o MVP. Observando a mediana, a arquitetura MVVM mostrou uma redução de 25,67% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

A Figura 16 mostra o desempenho de CPU para pesquisar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android Studio.

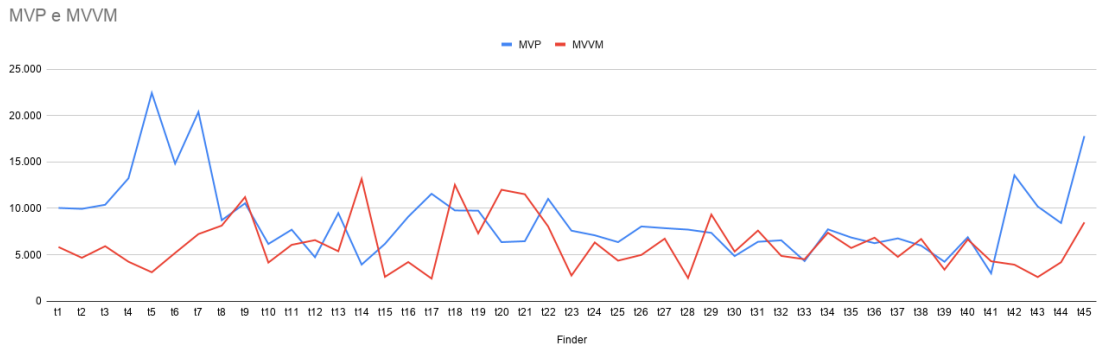

Figura 16 – Desempenho do uso CPU para pesquisar em microsegundos

O intervalo de 95% de confiança para o desempenho do uso de CPU em pesquisar tarefa no MVP varia entre [7.590; 9.945]. O intervalo de 95% de confiança para o desempenho do uso de CPU em pesquisar tarefa no MVVM varia entre [5.330; 6.925].

A Figura 17 mostra o uso de memória para pesquisar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta Profile do Android Studio.

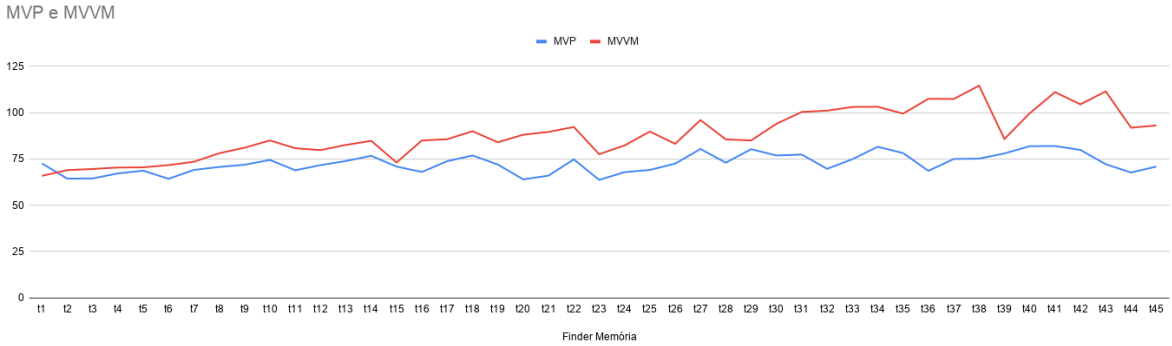

Figura 17 – Desempenho do uso de memória para pesquisar em MB

O intervalo de 95% de confiança para o desempenho do uso de memória em pesquisar tarefa no MVP varia entre [70.9; 74.0]. O intervalo de 95% de confiança para o desempenho do uso de memória em pesquisar tarefa no MVVM varia entre [84.7; 92.0].

A Tabela 11 mostra a média, mediana e desvio da execução em tempo real da função editar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes.

Observando a média presente na Tabela 11, o padrão arquitetural MVVM mostrou um aumento de 17.98% no tempo real do uso de memória em comparação com o MVP. Verificando a mediana, o padrão arquitetural MVVM mostrou um aumento de 15,75% no uso de memória em comparação com o MVP. Então a qualidade do desempenho foi MVP

Tabela 11 – Resultado do uso de memória para pesquisar filmes com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	69.8	72.5	69.1	72.2	5,17
MVVM	80.0	88.4	81.1	85.7	12,6

> MVVM, a qualidade do desempenho do uso de memória com o MVP foi melhor neste caso.

6.1.3 Avaliação de desempenho

O desempenho é avaliado pela medição de tempo, foi análisado o tempo de execução real das funções até o seu termino. O MVVM apresentou um melhor resultado para a métrica de desempenho dos testes em tempo real de execução, mas em algumas funções como listar e criar a diferença não foi tão significativa, do ponto de vista de memória na função pesquisar o MVVM teve um aumento de consumo. O ambiente de teste foi um dispositivo Android com um hardware da atualidade e ultima versão da API. Os resultados podem variar se forem aplicados em dispositivos diferentes.

7 RESUMO DOS RESULTADOS

Esta seção é responsável por apresentar um resumo dos resultados. O padrão arquitetural que apresentar uma execução no tempo real em menor tempo, ou um menor uso de memória, então deve indicar um melhor desempenho.

7.1 Desempenho do TODO

1. Criação da tarefa - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP < MVVM
    - Intervalo de confiança para CPU no MVP: [12.681; 15.854]
    - Intervalo de confiança para CPU no MVVM: [10.202; 14.008]
    - Intervalo de confiança para memória no MVP: [78.01; 80.18]
    - Intervalo de confiança para memória no MVVM: [75.41; 77.98]
2. Deletar a tarefas - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP < MVVM
    - Intervalo de confiança para CPU no MVP: [9.359; 12.115]
    - Intervalo de confiança para CPU no MVVM: [6.442; 7.213]
    - Intervalo de confiança para memória no MVP: [89.4; 96.9]
    - Intervalo de confiança para memória no MVVM: [83.2; 86.9]
3. Listar Tarefas - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP < MVVM
    - Intervalo de confiança para CPU no MVP: [9.256; 11.342]
    - Intervalo de confiança para CPU no MVVM: [8.312; 11.613]
4. Editar tarefa - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP < MVVM
    - Intervalo de confiança para CPU no MVP: [15.852; 17.237]

- Intervalo de confiança para CPU no MVVM: [12.309; 14.848]
- Intervalo de confiança para memória no MVP: [81.3; 85.6]
- Intervalo de confiança para memória no MVVM: [78.3; 81.5]

7.2 Desempenho do Finder

1. Pesquisar tarefa - Aplicação *Finder*
    - CPU: MVP < MVVM
    - Memória consumida: MVP > MVVM
    - Intervalo de confiança para CPU no MVP: [7.590; 9.945]
    - Intervalo de confiança para CPU no MVVM: [5.330; 6.925]
    - Intervalo de confiança para memória no MVP: [70.9; 74.0]
    - Intervalo de confiança para memória no MVVM: [84.7; 92.0]

Os resultados mostram que a arquitetura MVVM pode ter um melhor desempenho do uso de CPU. O desempenho do uso de memória para o MVVM foi melhor em quase todos os casos, mas na função pesquisar o MVVM apresentou um consumo de memória maior quando comparado com o MVP.

8 CONCLUSÃO E TRABALHOS

Este trabalho tem como objetivo definir se o padrão arquitetural MVVM com os cenários levantados pode ser melhor que o MVP que é mais conservador no mercado. Para encontrar os resultados foram preciso estudar conceitos básicos do padrão arquitetural, sua estrutura e definir semelhanças e diferenças.

O tempo necessário para executar suas funções e o uso de memória são duas medidas para mostrar que existe um desempenho melhor. Executando as aplicações testadas com automação para garantir que os fatores com inserção de textos e cliques sejam os mesmos, em seguida, foi utilizado uma ferramenta de desempenho do Android Studio para medir o tempo de execução real de cada uma das funções presentes.

O MVVM apresentou os melhores resultados em termos do desempenho de CPU. Do ponto de vista de memória, apenas na pesquisa apresentou um consumo mais alto de memória. Os casos de uso do padrão arquitetural MVP é necessário mais tempo para ser executado e em alguns casos necessita de mais memória.

Na avaliação de desempenho, o padrão arquitetural MVVM teve os melhores resultados nas métricas de CPU e memória, ou seja, o MVVM leva um melhor desempenho do que o padrão MVP, levando em consideração os cenários e o aparelho que foi usado.

8.1 Trabalhos futuros

Durante o desenvolvimento deste trabalho foi percebido pontos que podem ser melhorados e novos pontos para acrescentar na pesquisa que são de interesse na comunidade do Android, são eles:

- Aumentar o número de amostras para obter um intervalo de confiança melhor.
- Levantar mais cenários que poderia dar melhores resultados e ter um noção maior sobre qual padrão arquitetural pode abranger mais cenários que simulam os aplicativos nos dias de hoje.
- Obter aplicações mais complexas implementadas com diferentes arquiteturas, mas com mesma função.
- A pesquisa poderia incluir métricas de capacidade de teste e modificabilidade para comparar.
- Utilizar uma maior variedade de dispositivos Android para saber se o tipo de aparelho influência no desempenho.

- Investigar padrões arquiteturais FLUX e VIPER para poder encontrar melhores resultados. Novas arquiteturas estão surgindo ao longo dos anos e comparar elas com arquiteturas tradicionais pode criar uma pesquisa forte com grandes resultados na comunidade.

REFERÊNCIAS

[1] Android - Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia .org/wiki/Android.>. Acesso em: 19 ago. 2019.

[2] LINHARES. Paulo. Android Architecture Components (Componentes de Arquitetura Android). 2018. Disponível em:< https://medium.com/iterative/android-architecture-components-componentes-de-arquitetura-android-6c6be539f47e>. Acesso em: 19 ago. 2019.

[3] LOU, T. A comparison of Android Native App Architecture MVC, MVP and MVVM. 2016. Disponível em:<https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou_2016.pdf>. Acesso em: 19 ago. 2019.

[4] HUMENIUK, Vladyslav. Android Architecture Comparison: MVP vs. VIPER. 2019. Disponível em:<http://lnu.diva-portal.org/smash/get/diva2:1291671/FULLTEXT01.pdf>. Acesso em: 19 ago. 2019.

[5] Android Developers, 2019. Disponível em:<https://developer.android.com/docs>. Acesso em: 19 ago. 2019.

[6] Activity | Android developers. 2019. Disponível em: <https://developer.android .com/reference/android/app/Activity>. Acesso em: 19 ago. 2019.

[7] CICLO DE VIDA DE UN ACTIVITY - ANDROID STUDIO. Disponível em: <https://www.youtube.com/watch?v=nv_J-VGnQVo>. Acesso em: 19 ago. 2019.

[8] Fragmentos | Android developers. 2019. Disponível em: <https://developer .android.com/guide/components/fragments?hl=pt_br>. Acesso em: 19 ago. 2019.

[9] MVP Android. 2017. Disponível em: <https://www.thiengo. com.br/mvp-android>. Acesso em: 19 ago. 2019.

[10] NUNES, Felipe. Android MVC x MVP x MVVM qual Pattern utilizar — Parte 1. 2017. Disponível em:<https://medium.com/@FilipeFNunes/android-mvc-x-mvp-x-mvvm-qual-pattern-utilizar-parte-1-3defc5c89afd>. Acesso em: 19 ago. 2019.

[11] Model-View-Presenter - Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Model-view-presenter>. Acesso em: 19 ago. 2019.

[12] Cadu. Entendendo o pattern Model View ViewModel MVVM. 2010. Disponível em: <https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>. Acesso em: 19 ago. 2019.

[13] CAVALCANTE, Henri. Flux — A arquitetura do Facebook para desenvolvimento FrontEnd. 2015. Disponível em:<https://medium.com/@henricavalcante/flux-a-arquitetura-do-facebook-para-desenvolvimento-frontend-2bf7192c8f77>. Acesso em: 19 ago. 2019.

[14] Flux Architecture on Android. 2015. Disponível em: <https://github.com /lgvalle/android-flux-todo-app>. Acesso em: 19 ago. 2019.

[15] CHUGH, Anupam. Android MVVM Design Pattern - JournalDev. 2018. Disponível em:<https://www.journaldev.com/20292/android-mvvm-design-pattern>. Acesso em: 19 ago. 2019.

[16] Profile your app performance | Android Developers. 2019. Disponível em: <https://developer.android.com/studio/profile/>. Acesso em: 19 ago. 2019.

[17] CARVALHO, Bruno. Appium: automação para apps mobile - assert(QA) - Medium. 2018. Disponível em:<https://medium.com/assertqualityassurance/appium-automação-para-apps-mobile-a256db64a27d>. Acesso em: 19 ago. 2019.

[18] Tutorial para execução dos testes, resultados e amostras. Disponível em: <https://github.com/alexandrefcesar/tcc-codigo-amostras>. Acesso em: 19 ago. 2019.