UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Victor Edmond Freire Gaudiot

Análise de arquiteturas de módulos em projetos MVVM

Recife

2025

Victor Edmond Freire Gaudiot

ANÁLISE DE ARQUITETURAS DE MÓDULOS EM PROJETOS MVVM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Orientador (a): Paulo Henrique Monteiro Borba

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Gaudiot, Victor Freire.
Análise de arquiteturas de módulos em projetos MVVM / Victor Freire Gaudiot. - Recife, 2025.
17 p.

Orientador(a): Paulo Monteiro Borba
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
Inclui referências.

1. Arquitetura de Módulos. 2. MVVM. 3. Desenvolvimento de Software. 4. Experiência do Desenvolvedor. I. Borba, Paulo Monteiro. (Orientação). II. Título.

000 CDD (22.ed.)

Victor Edmond Freire Gaudiot

Análise de arquiteturas de módulos em projetos MVVM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: 04/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Henrique Monteiro Borba (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Paulo Henrique Monteiro Borba (Examinador Interno)

Universidade Federal de Pernambuco

_____

Profa. Dra. Paola Rodrigues de Godoy Accioly (Examinadora Interna)

Universidade Federal de Pernambuco

Análise de arquiteturas de módulos em projetos MVVM

Victor Gaudiot

Centro de Informática

Universidade Federal de Pernambuco

Brasil

vefg@cin.ufpe.br

Resumo

No desenvolvimento de software é comum que os programadores respeitem princípios como SOLID, YAGNI e DRY, além de utilizar um padrão arquitetural como Clean, Hexagonal e MVVM. Essas decisões, além de terem impacto no produto final, são fundamentais para aprimorar a experiência do desenvolvedor (DX), e desta forma facilitar a implementação e manutenção de funcionalidades. Entretanto, a literatura e as práticas de mercado pouco exploram como deve ser a arquitetura de módulos - que define a organização de módulos, submódulos e arquivos - em uma aplicação. A falta de cuidado ao escolher uma arquitetura pode trazer consequências negativas para o projeto, como duplicação acidental de código, dificuldade de navegação entre arquivos e maior tempo de adaptação de novos colaboradores. Este artigo propõe e avalia duas arquiteturas de módulos aplicadas a um mesmo projeto *toy*. São realizadas seis modificações no sistema de cada arquitetura, abrangendo tanto mudanças funcionais quanto não-funcionais, para avaliar o impacto na DX após cada alteração. Os resultados indicam que a escolha da arquitetura de módulos deve levar em consideração o escopo e o porte do projeto, bem como os tipos de mudanças mais frequentes.

Palavras-chave: Arquitetura de Módulos, MVVM, Desenvolvimento de Software, Ex-

periência do Desenvolvedor.

1 Introdução

Os engenheiros de software modernos trabalham em projetos que possuem milhares de linhas de código e continuam a crescer com a adição de novas funcionalidades, refatorações e manutenções. Esse constante aumento torna o gerenciamento da base de código uma tarefa cada vez mais complexa. Para minimizar esse problema é de costume que os programadores adotem práticas para melhorar a experiência do desenvolvedor[7](DX, do acrônimo em inglês), tais como os princípios SOLID[15], YAGNI[9] e DRY[12], assim como um padrão arquitetural, podendo ser Clean[16], Hexagonal[17], MVVM[2]. Esses padrões, além de terem impacto no produto final[20], auxiliam um processo de desenvolvimento mais eficiente, robusto e direcionado, acelerando a entrega e manutenção de funcionalidades[4].

Como mostrado por Bergman et al.[3] na vida pessoal, mesmo que um usuário possa agrupar todos os documentos pessoais em uma única pasta, é natural que esse desenvolva um sistema de organização para juntar documentos de significados similares, assim tornando-os mais fáceis de localizar caso necessários. No ambiente profissional, projetos de software são elaborados por equipes de diversos tamanhos[14, 11], onde cada colaborador, influenciado pela sua vida privada, tem uma preferência em como determinar a estrutura de pastas do projeto no qual está contribuindo, o que nem sempre resulta em consenso dentro da equipe.

A arquitetura de módulos[1] - que neste trabalho é intitulada de "modulação" - refere-se à maneira como uma equipe estrutura os diretórios e arquivos[2] de um projeto em desenvolvimento. Embora diferentes abordagens de modulação já sejam aplicadas no mercado, sua adoção ocorre de forma não padronizada, sendo definida pelos desenvolvedores sem critérios bem estabelecidos. Essa situação provavelmente se deve à pouca literatura que oriente a escolha de uma modulação adequada para aprimorar a experiência do desenvolvedor. Como consequência, diferentes equipes adotam arquiteturas de módulos variadas, sem a existência de um padrão consolidado, ao contrário do que ocorre com padrões arquiteturais, como Clean, Hexagonal e MVVM.

A escolha de uma modulação bem estruturada pode trazer diversos benefícios para a equipe de desenvolvimento, incluindo maior agilidade na criação e manutenção de funcionalidades, facilidade na navegação pelo projeto e redução do tempo de adaptação de novos

[1] Neste estudo, os termos pasta, módulo e diretório serão utilizados como sinônimos.
[2] Neste estudo, os termos arquivo e documento serão utilizados como sinônimos.

colaboradores.

Diante disso, este estudo propõe duas possíveis arquiteturas de módulos para projetos MVVM: MVVM-Default e Feature-Based - que serão descritas mais adiante. Também tem o objetivo de compreender quais aspectos devem ser considerados pelos desenvolvedores ao optar entre uma delas.

2 Arquiteturas de módulos estudadas

A arquitetura de módulos, ou modulação, refere-se à organização dos diretórios de um projeto, e dentro de quais desses cada arquivo que compõe o sistema será armazenado.

No contexto deste artigo, **src** refere-se ao módulo raiz que contém os principais módulos responsáveis pela implementação das funcionalidades da aplicação. Ele serve como o ponto central de organização do código-fonte, agrupando os submódulos diretamente relacionados ao desenvolvimento das *features* do sistema.

Neste estudo, focamos apenas nos submódulos diretos de **src**, denominados aqui como **"Módulos principais"**. Os demais módulos, que se mantêm inalterados entre as arquiteturas analisadas, não serão abordados, pois são destinados à configuração do projeto. Entre eles, incluem-se módulos responsáveis por conexão com o banco de dados, gerenciamento de bibliotecas de terceiros e *assets*.

Neste estudo as arquiteturas propostas são a MVVM-default, baseado no conceito do padrão arquitetural MVVM, e o Feature-Based, inspirado no conceito de slices abordado por Dias et al. [6].

2.1 MVVM-Default

A arquitetura MVVM-Default realiza o agrupamento dos arquivos de funcionalidades em três módulos principais: "Model", "View" e "ViewModel". Cada um desses diretórios representa uma camada do padrão arquitetural MVVM. Sendo uma abordagem simples, direta e intuitiva.

A figura 1 exemplifica essa modulação, onde o módulo "models" agrupa os arquivos relacionados à camada de dados, indicados pela extensão ".model.dart". O diretório "views" contém os documentos da camada de UI, com extensão ".view.dart", e o módulo "viewmodels" detém os arquivos da camada lógica, simbolizados pela extensão ".viewmodel.dart".

2.2 Feature-Based

A arquitetura Feature-Based organiza o projeto em módulos principais que representam funcionalidades específicas da aplicação. Essa abordagem facilita a separação de responsabilidades, proporcionando ao desenvolvedor uma visão clara da estrutura do projeto e facilitando a identificação e gerenciamento de arquivos relacionados a uma mesma funcionalidade.

A figura 2 demonstra essa modulação. O módulo ***"car"*** junta os arquivos relacionados à funcionalidade de carros, representados pelo início "*car*". O diretório ***"motorcycle"*** agrupa os documentos da funcionalidade de motocicletas, indicados pelo início "*motorcycle*", e a pasta ***"user"*** que contém os arquivos da funcionalidade de usuários, simbolizado pelo início "*user*".

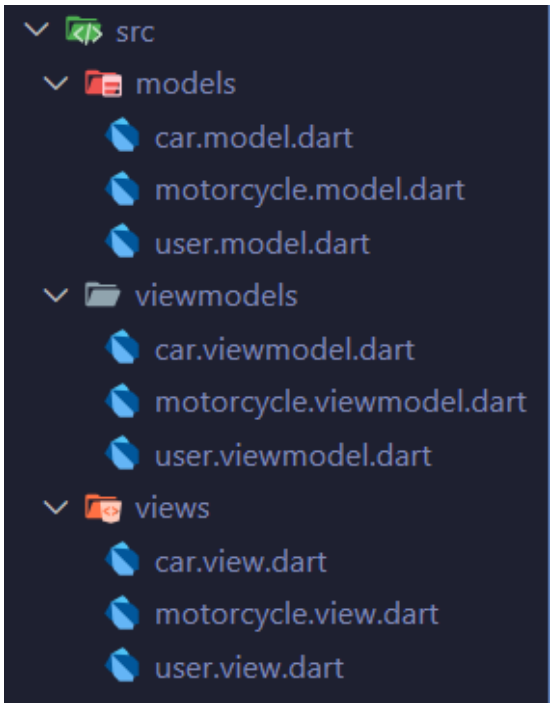

Figura 1: Exemplo da arquitetura MVVM-Default.

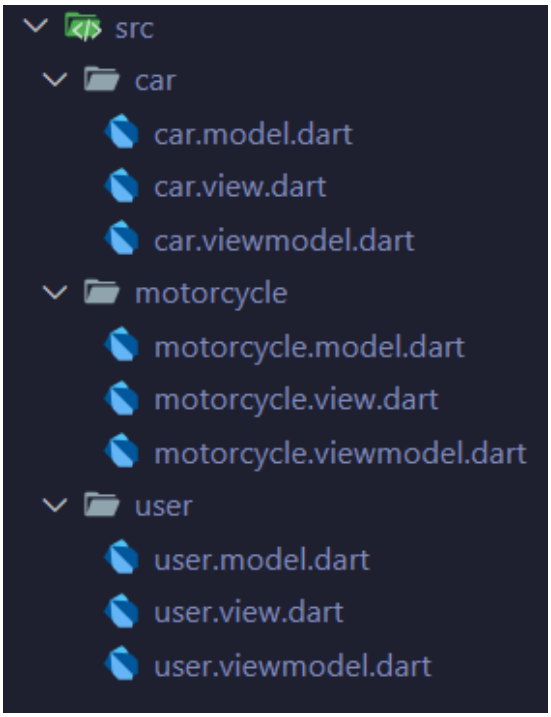

Figura 2: Exemplo da arquitetura Feature-Based.

3 Metodologia

Neste estudo é investigado como uma arquitetura de módulos pode impactar aspectos de DX como facilidade de navegação entre arquivos. Para isso usamos um projeto *toy*, um aplicativo *mobile* aplicando o padrão arquitetural MVVM. Também assumimos que boas práticas populares estão sendo respeitadas, como os princípios SOLID, KISS e DRY, nomenclatura de pastas e arquivos e separação de arquivos nas suas responsabilidades únicas. Para conduzir este estudo,

realizamos seis *Change Requests* (CRs), cada uma planejada para impactar um aspecto específico do sistema.

Foram utilizadas quatro métricas para verificar como a modulação pode colaborar na organização, navegabilidade e gerenciamento dos módulos e documentos. Duas delas avaliam a estrutura de pastas completa da aplicação, e as outras verificam apenas os diretórios que sofreram alteração na *Change Request*.

3.1 Escolha do projeto *toy*

O projeto *toy* deste estudo é um aplicativo *mobile* (app) de gerenciamento de filmes desenvolvido com Flutter[10] e no padrão arquitetural MVVM. Nele o usuário pode buscar filmes e salvar em uma lista aqueles que possui interesse de assistir futuramente, além de sinalizar quais filmes já foram assistidos. Os dados dos filmes são obtidos via a API do TMDB[19] e armazenados localmente no dispositivo do usuário. A escolha desse sistema se deve ao fato de já ter sido previamente desenvolvido pelo autor, o que permitiu concentrar os esforços na organização modular do projeto, sem a necessidade de construir uma aplicação do zero.

Este estudo também é válido para outras linguagens de programação e frameworks, visto que em nenhum momento é abordado peculiaridades do Flutter. Foi escolhido utilizar MVVM visto que sua criação já foi pensada para trabalhar junto à apps, e advém como uma evolução para resolver problemas de acoplamento entre as camadas dos padrões MVC e MVP.

3.2 Escolha do padrão arquitetural MVVM

O *Model-View-ViewModel* (MVVM) é um padrão arquitetural de software que promove a separação entre a camada de dados, a interface do usuário (UI) e a lógica de negócios, facilitando a manutenção e escalabilidade da aplicação. Essa abordagem foi originalmente proposta por arquitetos da Microsoft, sendo amplamente adotada no desenvolvimento de aplicativos móveis e sistemas frontend modernos.

A principal diferença entre MVVM e *Model-View-Controller* (MVC) está na maneira como a UI se comunica com a lógica da aplicação. No MVC, a *View* interage diretamente com o *Model*, o que pode levar a um acoplamento maior entre essas camadas. Já no MVVM, a camada intermediária *ViewModel* atua como um mediador, garantindo que a *View* não tenha acesso direto ao *Model*. Isso promove um maior desacoplamento, e permite a reutilização de componentes, maior testabilidade e uma separação mais clara das responsabilidades.

A escolha de utilizar MVVM para este estudo se deve ao fato de representar uma evolução do MVC, e pensada em aplicações *mobile*. Sendo projetada especificamente para aplicações que exigem uma separação bem definida. Além disso, a utilização do *ViewModel* reduz a dependência entre as camadas, permitindo maior modularidade e facilitando a manutenção do código. Essas características tornam o MVVM uma abordagem ideal para a análise de diferentes arquiteturas de módulos neste artigo.

3.3 Evolução do projeto

O projeto parte de uma implementação inicial onde está configurado acesso à API, armazenamento local e três telas, sendo elas a de buscar filmes (figura 3), lista de filmes de interesse (figura 4) e lista de filmes assistidos (figura 5). Dentro do mesmo repositório implementamos a arquitetura MVVM-Default e Feature-Based em diferentes diretórios (pastas lib_mvvm e lib_feature, respectivamente), de forma que possam compartilhar pastas em comum (como *base* e *shared*), mas garantido que não importem documentos um do outro (figura 6).

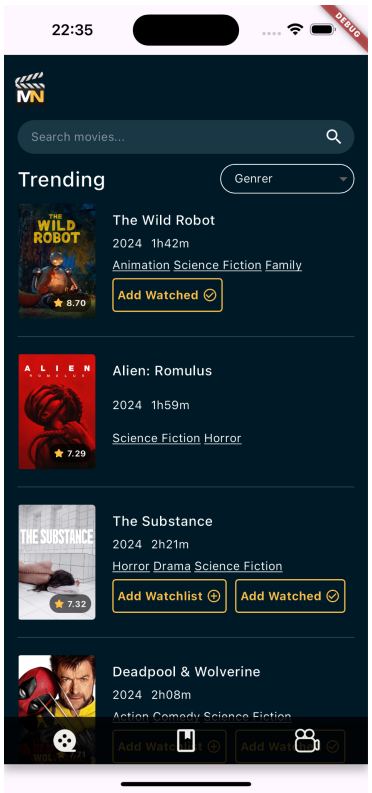

Figura 3: Tela de busca.

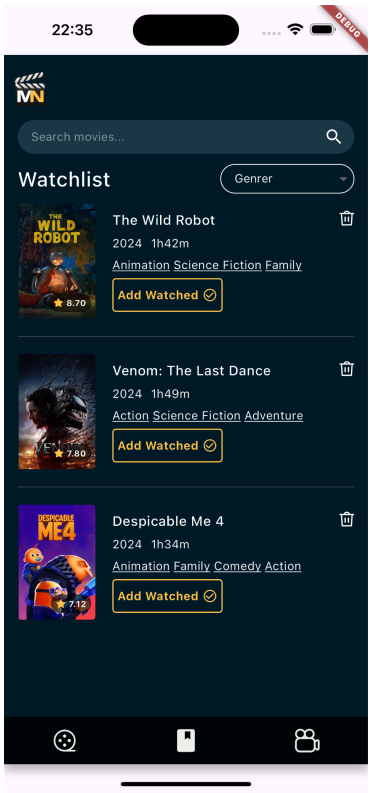

Figura 4: Tela de interesse.

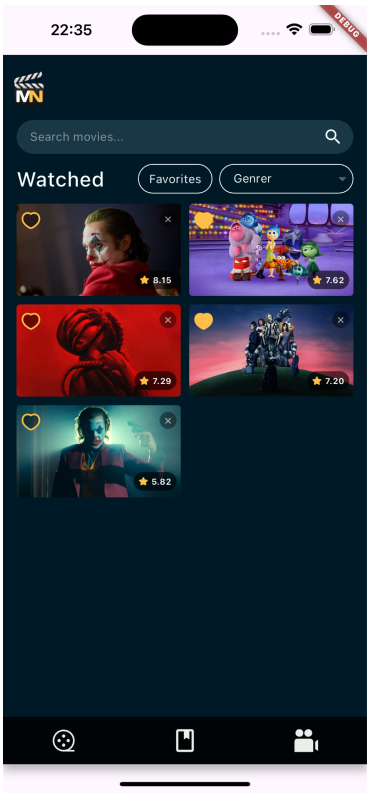

Figura 5: Tela de assistido.

A implementação de cada CR foi realizada inicialmente na arquitetura MVVM-Default

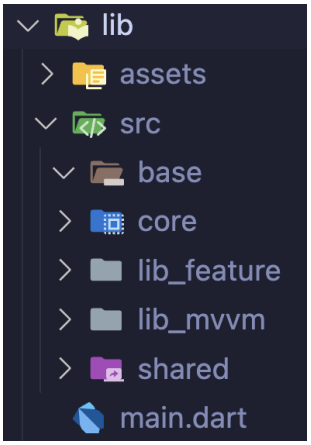

Figura 6: Ambas arquiteturas podem acessar recursos de base, core e shared, mas não interagem uma com a outra.

e, posteriormente, na Feature-Based.[3] Ao término de cada modificação, as métricas, sendo definidas mais à frente, foram calculadas.

Um aspecto relevante deste estudo é que, como o conteúdo dos arquivos permanecem idênticos em ambas as arquiteturas de módulos, a única diferença entre as modulações está nos módulos, submódulos e na estrutura dos documentos dentro do projeto.

As seguintes *Change Requests* foram implementadas, onde também descrevemos como ela impactou o sistema:

1. Uma nova funcionalidade permitindo que o usuário possa acessar uma tela com mais detalhes do filme. (nova *feature*)

2. Alterar a biblioteca de armazenamento local de Shared Preferences [8] para Hive [13]. (requisito não-funcional)

3. Alteração na camada View para incluir um esqueleto de carregamento com efeito shimmer enquanto o conteúdo da tela não está pronto. (nova *feature*)

4. Modificar a camada Model para que a API receba os dados dos filmes em pt-BR. (*bug fix*)

5. Adicionar à tela de detalhes do filme a listagem de streamings que disponibilizam o filme selecionado. (nova *feature*)

6. Alteração na camada ViewModel para que a ordenação dos filmes seja feito pela avaliação dele ao invés do tempo de duração. (*bug fix*)

[3] O código-fonte da implementação final das arquiteturas está disponível em: https://github.com/Gaudiot/MovieNight_tcc/tree/main2-cr6.

É possível ver que a partir no nosso sistema inicial (indicado por TS) realizamos a implementação da CR1 no sistema que possui a arquitetura MVVM-Default (indicado por TS-M CR1), e em seguida refletimos as alterações na arquitetura Feature-Based (indicado por TS-F CR1). Com a alteração feita, calculamos as métricas e após repetimos o clico com a implementação da próxima CRi a partir do projeto TS-F CR(i-1) até que sejam finalizadas as alterações propostas. O fluxo dessa evolução pode ser visto na figura 7.

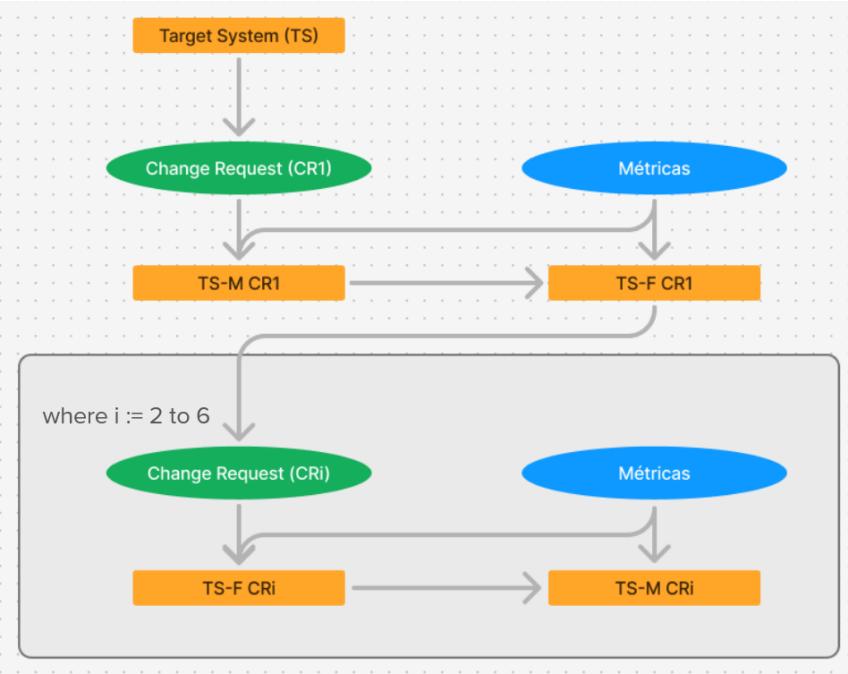

Figura 7: Fluxo da evolução do projeto.

3.4 Métricas utilizadas

Neste estudo são definidas quatro métricas. Em duas delas é realizada uma análise global da estrutura de pastas, considerando todos os módulos principais, seus submódulos e arquivos. Nas outras há uma visão mais atômica, focada nas partes do código onde houve mudanças. Essas métricas avaliam se a arquitetura está organizada e modularizada, conforme abordado em outros estudos[3, 18].

Bergman et al. [3] definem a métrica *Module Size* (MS), para definir quantos itens (subpastas e documentos) uma pasta contém, e determinam 22,46 como um valor padrão para uso em computador pessoal. Apesar de ser uma métrica interessante para este estudo e estudos futuros, foi optado por não documentar os resultados visto que seu valor sempre ficou consideravelmente abaixo da referência e não traria conclusões relevantes já que valores inferiores não são

necessariamente negativos ou positivos.

Para auxiliar na definição das métricas, adotamos a notação $\delta(i, j)$ para indicar a quantidade de diretórios para navegar entre um diretório **i** e **j**[4], um exemplo pode ser visto na figura 8. A notação $\alpha(i)$ representa o diretório no qual se encontra o arquivo **i**.

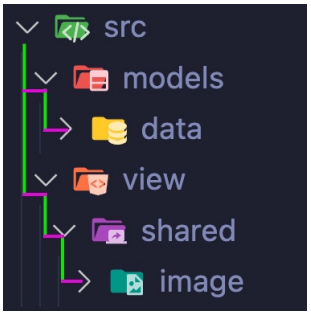

Figura 8: O valor de $\delta(data, image)$ é igual a 5, visto que o fluxo de navegação é data → models → src → view → shared → image.

3.4.1 Main Module Modification Factor (MMMF)

A métrica *Main Module Modification Factor* (MMMF), ou Fator de Modificação de Módulo Principal, verifica se as modificações realizadas em uma CR ficam centralizadas em um ou mais módulos principais.

A variação de modificação Δ_i determina o quanto o programador mudou de contexto dentro do módulo principal **i**.

$$\Delta_i = \frac{1}{m}\sum_{j=1}^{m}(1 - 0,5^{\delta(i,\alpha(j))})$$

onde **j** é o documento alterado e **m** a quantidade de documentos alterados do módulo principal.

$$MMMF = \sum_{i=1}^{n}\Delta_i = \sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m}(1 - 0,5^{\delta(i,\alpha(j))}) = \frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{m}(1 - 0,5^{\delta(i,\alpha(j))})$$

onde **n** é a quantidade de módulos principais que sofreram alguma modificação.

Como módulos principais distintos possuem contextos diferentes, valores de MMMF mais próximos de 1 indicam menor necessidade de mudança de contexto por parte do desenvolvedor, favorecendo a compreensão do código.

[4]Outro exemplo, se forem o mesmo diretório, o valor é 0. Caso j seja submódulo direto de i, o valor é 1.

Para exemplificar considere o app após a implementação da terceira *Change Request* na arquitetura Feature-Based. A Tabela 1 apresenta a variação de modificação de cada módulo principal alterado.

Com esses valores, é calculado um MMMF de 3,06.

X	home	profile	total
Δ_i	1,56	1,5	3,06

Tabela 1: Quantidade de arquivos por módulo principal

3.4.2 Import Depth Average (IDA)

Na implementação ou alteração de uma funcionalidade é comum que sejam realizadas importações de diversos arquivos auxiliares, muitas vezes em um diretório diferente do arquivo sendo modificado. A métrica *Import Depth Average* (IDA), ou Profundidade Média das Importações, ajuda a compreender o quanto em média o colaborador navega do arquivo **X** para o importado **Y**.

Para que o IDA possa analisar o comportamento do projeto ao fim de uma CR, serão consideradas apenas as novas importações realizadas em documentos modificados, desde que tanto a importação quando o documento pertençam a um módulo principal. Além disso, será contabilizado apenas uma única ocorrência nos casos que hajam múltiplos arquivos sendo importados de um mesmo diretório.

O fator de importação Γ_i simboliza quanto o desenvolvedor precisa navegar para realizar as novas importações do arquivo **i**.

$$\Gamma_i = \sum_{j=1}^{m} \delta(\alpha(i), \alpha(j))$$

onde **m** é a quantidade de novas importações do arquivo criado ou modificado e **j** o arquivo sendo importado.

$$IDA = \frac{1}{n} \sum_{i=1}^{n} \Gamma_i \ = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} \delta(\alpha(i), \alpha(j))$$

onde **n** é a quantidade de arquivos criados ou modificados que tiveram novas importações.

Valores próximos de zero indicam que arquivos de mesmo contexto estão em diretórios próximos, facilitando a navegação e manutenção do código.

Para exemplificar tome o projeto após a implementação da terceira *Change Request* na arquitetura MVVM-Default. A Tabela 2 apresenta o fator de importação de cada arquivo

alterado.

Com esses valores, é calculado um IDA de 0,5.

extensão	profile	search_movies	watchlist	watched
.view.dart	1	1	1	1
.skeleton.dart	0	0	0	0

Tabela 2: Quantidade de arquivos por módulo principal

3.4.3 Module Size Uniformity Index (MSUI)

Sarkar et al. [18] definem a métrica *Module Size Uniformity Index* (MSUI), ou Índice de Uniformidade do Tamanho do Módulo, para avaliar a distribuição de chamadas de funções externas. Neste estudo, adaptamos o MSUI para avaliar o grau de equilíbrio dos módulos principais, isto é, se os arquivos estão igualmente distribuídos entre as pastas.

$$MSUI = \frac{\mu}{\mu + \sigma}$$

onde μ é a média da quantidade de arquivos por módulo principal e σ é o desvio padrão.

Valores de MSUI próximo de 1 indicam uma organização mais uniforme, o que pode reduzir o tempo médio para o desenvolvedor localizar um documento dentro do projeto.

Para exemplificar tome o projeto após a implementação da primeira *Change Request* na arquitetura MVVM-Default. A Tabela 3 apresenta a quantidade de arquivos em cada módulo principal.

Com base nesses valores, é calculada uma média de 9,667 e um desvio padrão de 4,028, resultando em um MSUI de 0,706.

	Model	View	ViewModel
# de arquivos	13	12	4

Tabela 3: Quantidade de arquivos por módulo principal.

3.4.4 Mean Depth (MD)

Bergman et al. [3] utiliza a métrica *Mean Depth* (MD), ou Profundidade Média, para quantificar a profundidade dos arquivos pessoais recentemente acessados pelos usuários. O estudo mostra que, em média, um usuário leva 14,76 segundos para encontrar o documento desejado, correlacionando esse tempo com a profundidade dos arquivos na hierarquia do sistema.

Com base nos resultados do estudo de Bergman, também é adotado o valor de referência 2,86 para esta métrica. Porém, diferente do estudo original, nossa análise considera apenas o conjunto dos arquivos contidos em um módulo principal, permitindo avaliar a organização estrutural de uma modulação no contexto deste trabalho.

$$MD = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

onde X_i é a profundidade do arquivo **i** definida pela quantidade de diretórios entre o módulo principal que o contém e o diretório no qual se encontra.

Para exemplificar a métrica, tome novamente a implementação da primeira *Change Request* na arquitetura MVVM-Default. A tabela 4 disponibiliza a profundidade dos arquivos por módulo principal. O valor do MD para este caso é 1,655.

profundidade	Model	View	ViewModel	Total
1	0	6	4	10
2	13	6	0	19

Tabela 4: Exemplo de cálculo do *Mean Depth*

3.4.5 Objetivo das métricas

As métricas MMMF e IDA foram planejadas de forma a evidenciar os aspectos atômicos do desenvolvimento, ou seja, as partes em que o programador está efetivamente interagindo enquanto realiza alguma atividade. Enquanto as métricas MSUI e MD foram adaptadas a partir de outros estudos e visam esclarecer aspectos globais do sistema, portanto, como as pastas e arquivos ficam organizados e a facilidade de navegação e de encontrar arquivos desejados.

3.5 Resultados

Após a aplicação das modificações tanto na modulação MVVM-Default e na Feature-Based as métricas foram calculadas e podem ser visualizadas nas tabelas disponibilizadas. Os resultados de MMMF, IDA, MSUI e MD estão apresentados respectivamente nas tabelas 5, 6, 7 e 8.

E a implementação final do projeto pode ser encontrado no github.

Arquitetura	CR1	CR2	CR3	CR4	CR5	CR6
MVVM-Default	2,75	0,00	1,25	1,50	3,81	1,00
Feature-Based	2,37	0,00	3,06	1,50	2,75	1,50

Tabela 5: Resultados da métrica *Main Module Modification Factor*. Quanto maior o valor pior o resultado, onde o ideal é 1.

Arquitetura	CR1	CR2	CR3	CR4	CR5	CR6
MVVM-Default	3,00	0,00	0,50	0,00	3,17	0,00
Feature-Based	0,40	0,00	0,67	0,00	1,14	0,00

Tabela 6: Resultados da métrica *Import Depth Average*. Quanto maior o valor pior o resultado, onde o ideal é 0.

4 Conclusões

Os resultados deste estudo mostram que ambas as arquiteturas apresentam valores de *Mean Depth* (MD) e *Module Size* (MS) dentro das expectativas para modularização eficiente. No entanto, a métrica *Module Size Uniformity Index* (MSUI) indica que a arquitetura MVVM-Default distribui os arquivos de forma mais equilibrada entre os módulos principais, reduzindo a diferença no tamanho dos diretórios.

Ao analisar as Change Requests individualmente, observamos que a arquitetura Feature-Based apresentou melhores resultados nas CRs 1, 4 e 5, que envolvem criação e alteração de funcionalidades. Esse padrão sugere que essa abordagem pode ser mais adequada para projetos que estão em constante evolução funcional. Porém, a arquitetura MVVM-Default obteve melhor desempenho nas CRs 3, 4 e 6, que envolvem modificações em uma das três camadas do padrão arquitetural MVVM. Isso indica que essa arquitetura pode ser mais eficiente para projetos nos quais as *Change Requests* costumam abordar se concentrar na camada *Model* ou *View* ou *ViewModel*.

Além disso, os resultados mostram que em mudanças não relacionadas às funcionalidades, como a CR 2, não houve impacto significativo na escolha da arquitetura. Isso sugere que, para esse tipo de modificação, a estrutura de pastas do projeto não influencia diretamente a complexidade da alteração.

Apesar deste estudo não ter produzido dados suficientes para analisar os casos de projetos de grande porte, ou de escopo aberto e flexível (como os desenvolvidos por startups), os dados indicam que à medida que novas funcionalidades são criadas ou aprimoradas, mais arquivos são desenvolvidos. Com o crescimento do número de documentos no projeto, o princípio da casa dos pombos[1] sugere que as métricas ***Module Size*** e ***Mean Depth*** também aumentem, podendo

Arquitetura	CR1	CR2	CR3	CR4	CR5	CR6
MVVM-Default	0,706	0,706	0,683	0,683	0,707	0,707
Feature-Based	0,542	0,542	0,539	0,539	0,585	0,585

Tabela 7: Resultados da métrica *Module Size Uniformity Index*. Quanto menor o valor pior o resultado, onde o ideal é 1.

Arquitetura	CR1	CR2	CR3	CR4	CR5	CR6
MVVM-Default	1,65	1,65	1,70	1,70	1,69	1,69
Feature-Based	1,62	1,62	1,91	1,91	1,89	1,89

Tabela 8: Resultados da métrica *Mean Depth*. O valor ideal é 2,86. Valores maiores são piores, mas não se pode afirmar o mesmo para valores menores.

eventualmente extrapolar os valores de referências dentro da arquitetura de módulos MVVM-Default por se limitar a apenas três módulos principais. No entanto, isso não necessariamente ocorre para Feature-Based visto à sua flexibilidade de conter múltiplos módulos principais.

Pela simplicidade e facilidade de implementação, a arquitetura MVVM-Default se mostra uma escolha adequada para projetos voltados a estudos pessoais, aplicações com escopo fechado e bem definido ou de médio e pequeno porte. Por outro lado, a modulação Feature-Based se mostra mais adequada para projetos de grande porte ou com escopo aberto e flexível. Dessa forma, conclui-se que cada arquitetura de módulos possui cenários específicos de aplicação, cabendo à equipe de desenvolvimento escolher a abordagem mais apropriada com base nas necessidades do projeto.

Este estudo pode servir como base para pesquisas futuras que analisem projetos de diferentes portes, escopos e tamanhos de equipe, possibilitando outras conclusões.

5 Ameaças à validade

Esta seção discute as principais ameaças à validade deste estudo. A primeira se deve pelo projeto toy ser desenvolvido por um único desenvolvedor, logo tornando inviável avaliar o fator tamanho de equipe. Também pode enviesar como será criado os submódulos dos módulos principais, além da arquitetura Feature-Based possuir uma subjetividade do que é considerado uma funcionalidade, influenciando no que será um módulo principal.

Outro viés deste estudo é por sempre iniciar a implementação da Change Request pela arquitetura MVVM-Default, o que pode influenciar a forma que o desenvolvedor iria agir caso precisasse desenvolver primeiramente na Feature-Based.

Apesar do projeto do estudo ser um caso real e abordando diversas camadas de implementação, é apenas um caso das infinitas possibilidades de sistemas que podem ser implementados. Modificações nas *Change Requests* analisadas podem resultar em métricas e conclusões diferentes das obtidas neste estudo.

Em muitos casos práticos do mercado, as aplicações desenvolvidas são de escopo aberto e de grande porte, o que se difere do que foi estudado neste artigo. Portanto, apesar de ser um guia inicial, ainda é preciso de mais estudos para investigar esses casos.

Além disso, sistemas de grande porte frequentemente utilizam ferramentas avançadas para navegação no código, reduzindo o impacto de diferenças na arquitetura de módulos escolhida. Em ambientes de desenvolvimento modernas, como o Visual Studio Code, oferecem funcionalidade nativas para auxiliar o programador na navegação entre arquivos através de recursos como pesquisa rápida ao digitar parte do nome de um arquivo, e por atalhos (como o ctrl + clique) que redirecionam o usuário para o documento contendo a implementação do componente selecionado.

Um dos fatores que são analisados aqui é a complexidade da navegação do desenvolvedor pelos módulos e arquivos do projeto. Sabe-se que ao atuar em determinada aplicação por um período de tempo o colaborador se familiariza com a base de código e os caminhos para os arquivos mais acessados. Isso sugere que, a longo prazo, a familiaridade do desenvolvedor pode reduzir os efeitos negativos de uma organização modular subótima.

Agradecimentos

Gostaria de dedicar este estudo a todos e todas que me apoiaram e acreditaram em mim na minha vida e ao longo da minha jornada acadêmica e tornaram possível eu ser a pessoa que sou hoje. Agradeço especialmente à minha mãe por me ensinar a ter paciência para quebrar aos poucos os tijolos da vida e sempre buscar ser a mudança que quero ver no mundo, ao meu pai por me mostrar o valor do trabalho bem feito e constantemente reforçar que independente das minhas decisões sempre terei um lugar seguro para voltar, à minha irmã por ser a minha melhor amiga e dar seus melhores conselhos nos momentos que mais estive perdido e desorientado e à minha namorada por me mostrar o que é amor, carinho e compreensão, e me apoiar na realização de todos os meus sonhos.

Também dedico em homenagem à Lola, minha gata que sempre estará em minhas memórias e em meu coração.

Quero mencionar também o trabalho de d'Amorim et al[5]. que serviu de inspiração para este artigo.

Referências

[1] M. Ajtai. *Parity and the Pigeonhole Principle*, pages 1–24. Birkhäuser Boston, Boston, MA, 1990.

[2] Chris Anderson. *The Model-View-ViewModel (MVVM) Design Pattern*, pages 461–499. Apress, Berkeley, CA, 2012.

[3] Ofer Bergman, Steve Whittaker, Mark Sanderson, Rafi Nachmias, and Anand Ramamoorthy. The effect of folder structure on personal file navigation. *JASIST*, 61:2426–2441, 12 2010.

[4] Simon Brown. *Software Architecture for Developers*. Leanpub, 2015.

[5] Fernanda d'Amorim and Paulo Borba. Modularity analysis of use case implementations. *The Journal of Systems and Software*, pages 1012–1027, 2012.

[6] Klissiomara Dias, Paulo Borba, and Marcos Barreto. Understanding predictive factors for merge conflicts. *Information and Software Technology*, 121:106256, 2020.

[7] Fabian Fagerholm and Jürgen Münch. Developer experience: Concept and definition. In *2012 International Conference on Software and System Process (ICSSP)*, pages 73–77, 2012.

[8] flutter.dev. Shared Preferences Package, 2025. Acessado em: 11/03/2025.

[9] Neal Ford. *The Productive Programmer*. O'Reilly Media, Sebastopol, CA, 2008.

[10] Google. Flutter: Beautiful native apps in record time, 2025. [Online; accessed 11-Mar-2025].

[11] Marjan Heričko, Aleš Živkovič, and Ivan Rozman. An approach to optimizing software development team size. *Information Processing Letters*, 108(3):101–106, 2008.

[12] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Boston, MA, 10 1999.

[13] isar.dev. Hive Package, 2025. Acessado em: 11/03/2025.

[14] Nesma Keshta and Yasser Morgan. Comparison between traditional plan-based and agile software processes according to team size project domain (a systematic literature review). In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 567–575, 2017.

[15] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[16] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[17] Robin Nunkesser. Using hexagonal architecture for mobile applications. In *Proceedings of the 17th International Conference on Software Technologies - Volume 1: ICSOFT,*, pages 113–120. INSTICC, SciTePress, 2022.

[18] Santonu Sarkar, Girish Maskeri, and Avinash Kak. Api-based and information-theoretic metrics for measuring the quality of software modularization. *Software Engineering, IEEE Transactions on*, 33:14–32, 02 2007.

[19] The Movie Database (TMDB). Tmdb api documentation, 2024.

[20] Lloyd G. Williams and Connie U. Smith. Performance evaluation of software architectures. In *Proceedings of the 1st International Workshop on Software and Performance*, WOSP '98, page 164–177, New York, NY, USA, 1998. Association for Computing Machinery.