Processando o Saber
eISSN 2179-5150
FATEC Praia Grande · fatecpg.edu.br/revista
submetido: Jan/2023 · **aceito**: Mar/2023 · **publicado**: Jun/2023
DOI 10.5281/zenodo.14927759

Modularização de aplicativos iOS

Modularization of iOS applications

Matheus Francisco da Silva Lima Gomes
Fatec Praia Grande
matheus.gomes12@fatec.sp.gov.br

Simone Maria Viana Romano
Fatec Rubens Lara
simone.romano@fatec.sp.gov.br

Jonatas Cerqueira Dias
Fatec Praia Grande
jonatas.dias2@fatec.sp.gov.br

RESUMO

O presente artigo buscou por meio de uma pesquisa bibliográfica exploratória, apresentar método de estabelecimento de arquitetura modular para aplicativos em dispositivos móveis Apple "iOS", contribuindo como material teórico para a comunidade de tecnologia. A arquitetura modular, é uma abordagem presente na arquitetura de software que consiste no dimensionamento de partes de um sistema em subsistemas capazes de existir de forma independente e de forma flexível. O objetivo do presente artigo é abordar desafios presentes na consolidação de uma aplicação de arquitetura modular em projetos que necessitam de dimensionamento de partes do sistema, devido o crescimento do projeto e das equipes, abordando pontos a serem consideradas para implantar essa abordagem em seu sistema. O método utilizado para formular as conclusões foi de natureza dedutiva, uma vez que se baseou em observações prévias e conhecimentos derivados do repertório bibliográfico existente. No que tange a procedimentos técnicos, foram utilizadas pesquisas no Google Acadêmico e bibliografias. Conclui-se que a arquitetura modular oferece diversos benefícios para a produtividade no processo de desenvolvimento, oferece menor complexidade para manutenibilidade e composição de novos sistemas. Deve-se considerar a curva de aprendizado para conceber o novo modelo de desenvolvimento de sistemas, o custo para habilitar esse modelo, assim como o nível de granularidade de cada módulo, evitando a criação de módulos generalizados e complexos demais para se manter.

PALAVRAS-CHAVE: Arquitetura Modular; Dispositivos Móveis; Arquitetura de Software; iOS.

ABSTRACT

This article sought through an exploratory literature research, present a method for establishing a modular architecture for applications in Apple mobile devices "iOS", contributing as theoretical material to the technology community. The modular architecture is an approach in software architecture that consists in the dimensioning of parts of a system in subsystems capable of existing independently and in a flexible way. The objective of this article is to address the challenges present in the consolidation of a modular architecture application in projects that require the dimensioning of system parts, due to the growth of the project and the teams, addressing points to be considered to implement this approach in your system. The method used to formulate the conclusions was deductive in nature, since it was based on previous observations and knowledge derived from the existing bibliographic repertoire. In terms of technical procedures, Google Scholar searches and bibliographies were used. It is concluded that modular architecture offers several benefits for productivity in the development process, offers less complexity for maintainability and composition of new systems. One must consider the learning curve to conceive the new system development model, the cost to enable this model, as well as the level of granularity of each module, avoiding the creation of generalized modules that are too complex to maintain.

KEYWORDS*: Modular Applications; Mobile Devices; Software Architecture; iOS.*

INTRODUÇÃO

Há diversos padrões, e estruturas que visam a implementação de sistemas de forma mais rápida, menos suscetível a erros, de fácil entendimento dentre tantos outros requisitos fundamentais para a construção de aplicações consistentes e sustentáveis. A estrutura de aplicações monolítica, foi uma das estratégias mais utilizadas, essa abordagem apresenta diversos desafios para prover ao sistema sustentabilidade de longo prazo (FOWLER, 2017). Um aplicativo monolítico tem todas ou a maioria das funcionalidades em um único sistema, além de gerenciar os componentes em camadas ou bibliotecas internas. Tal abordagem apresentará pontos negativos se ou quando o aplicativo aumentar suas funcionalidades, apresentar a necessidade de escalabilidade, aplicar manutenção e apresentar a necessidade de dimensionamento. Caso o aplicativo inteiro seja dimensionado, isso não será realmente um problema. Ainda assim, na maioria dos casos, algumas partes do aplicativo são os pontos de redução que exigem dimensionamento, enquanto outros componentes são menos utilizados (MICROSOFT, 2022). Algumas maneiras de lidar com os efeitos colaterais de sustentar um sistema monolítico, como o aumento do número de integrantes nas equipes de desenvolvimento com a finalidade de manter a velocidade de entrega, garantir a qualidade nos sistemas, se provaram insuficientes para comportar a aplicações com essas características (MARTIN, 2019).

A ausência de um projeto devidamente delimitado em contextos, resultam em retrabalho, que podem ser nomeadas como tarefas “toil” (SRE GOOGLE, 2017), termo utilizado pela equipe de engenheiros de confiabilidade da corporação Google para definir trabalhos repetitivos que não agregam valor para o produto. Ao passo que um aplicativo necessita de uma camada de interface para o usuário (GARTNER, 2022), que se refere à interface gráfica do sistema e ações disponíveis para interação do usuário, gerenciar regras de negócio específicas, desenvolvimento seguro, alta performance, usabilidade, alta disponibilidade para execução de experimentos para cada tipo de usuário, deixa de ser de uma atividade trivial, para uma grandioso estresse dentro das empresas com necessidade de escalabilidade rápida e de dimensionamento encontramos então aplicativos que se tornaram monolitos difíceis de se manter (FOWLER, 2017).

O objetivo deste artigo é apresentar os principais desafios para estabelecer um sistema de arquitetura modular, apresentar cuidados arquiteturais que devem ser considerados para provisionar às empresas um aplicativo saudável e consistente para atender as necessidades do mercado, proporcionando ao usuário “designs” mais modernos, fluídos, adaptabilidade para execução de experimentos e ofertas, validação de testes com mais velocidade. As equipes são altamente impactadas quando seus sistemas possuem arquiteturas engessadas, alto nível de acoplamento de código, funcionalidades com dependências implícitas, impedindo a aplicação de evoluir rapidamente e que não tenha a necessidade de mexer em todo projeto sempre que for criar uma tela na jornada do usuário (ANDROID DEVELOPER, 2022).

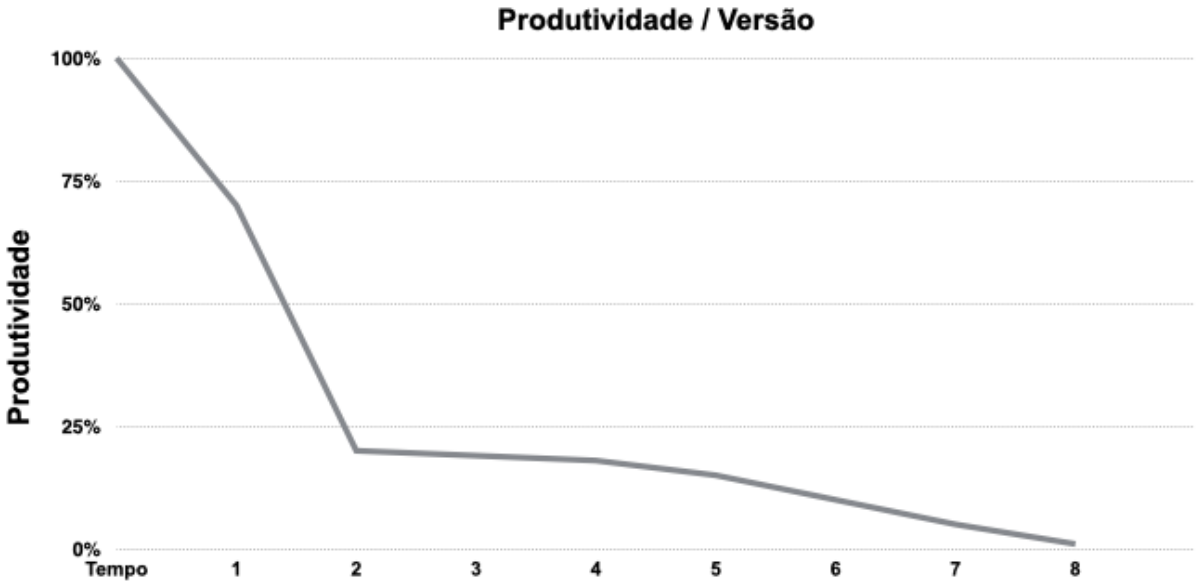
A modularização se faz necessária para viabilizar a entrega dos novos incrementos na aplicação, minimizando as responsabilidades da aplicação em pequenos contextos, reduzindo o tempo de desenvolvimento dos colaboradores e trazendo resultados significativos para a empresa. (OROSZ, 2021). É esperado que os times possam se dividir entregando novas funcionalidades enquanto outras equipes mexem na estrutura do projeto, os desafios técnicos são inúmeros como gestão de dependências, definição de padrões de arquitetura, separação das camadas do projeto a fim de expor as dependências de cada funcionalidade, integração dos módulos na camada principal do sistema, dentre outros desafios (VERNON, 2016). Diversos aspectos devem ser considerados no desenvolvimento de aplicações móveis, para garantir a sua escalabilidade, manutenibilidade e flexibilidade. Para delimitar os tópicos discutidos, o contexto dos aplicativos móveis destinados ao sistema operacional iOS é o foco do trabalho (OROSZ, 2021), que aborda diversos desafios e apresenta os modelos arquitetônicos e as características deste imenso ecossistema.

1. ARQUITETURA MODULAR

A arquitetura modular, consiste no dimensionamento de sistemas em subsistemas contidas e acopladas com flexibilidade (ANDROID DEVELOPER, 2022). Cada parte do sistema é definido como um módulo, que deve ser independente e deter finalidade clara. Tal abordagem se faz necessária quando uma aplicação se torna grande demais e os times também, gerenciar esse código em uma única aplicação aumenta o retrabalho, reduz a velocidade e influencia muito na organização (OROSZ, 2021).

A má constituição de uma arquitetura custa caro em produtividade (FOWLER, 2020). As entregas são cada vez mais vagarosas, mais caras e menos relevantes. Uma arquitetura bem planejada, proporciona mais produtividade e maior qualidade em seu sistema (MARTIN, 2019). Na figura 1, há um comparativo de produtividade de equipes que possuem um sistema com arquitetura mal implementada. O processo de análise considerou a quantidade de itens desenvolvidos dividido pelo número de lançamento de versões.

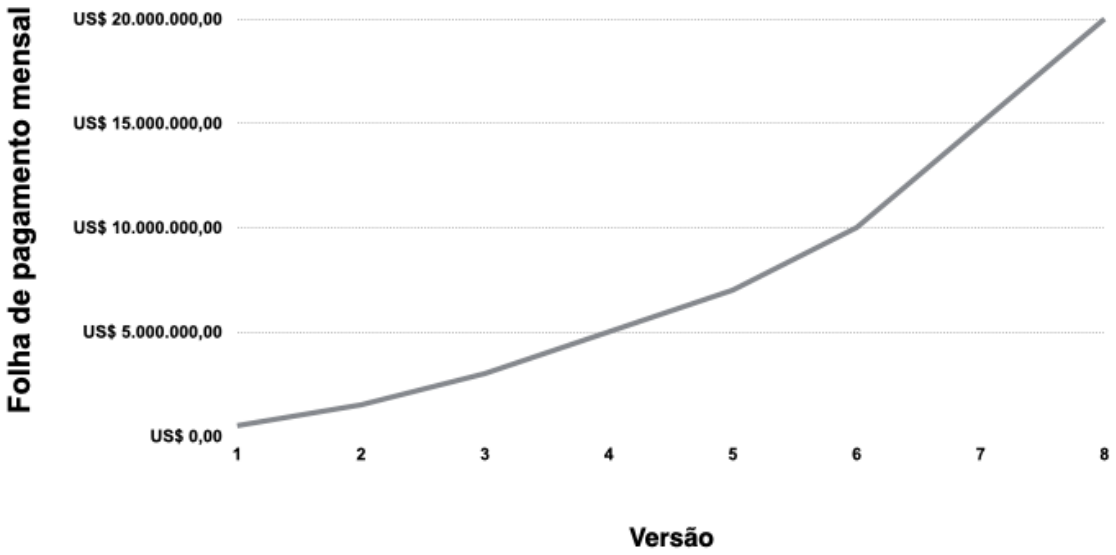
Figura 1 - Gráfico de produtividade sobre entregas das equipes



Fonte: Adaptado de (MARTIN, 2019)

A figura 2, segue-se a mesma análise, mas agora comparando o custo da empresa para cada lançamento de versão “release”, o que prova que uma arquitetura mal projetada, gera custo muito maior para as empresas.

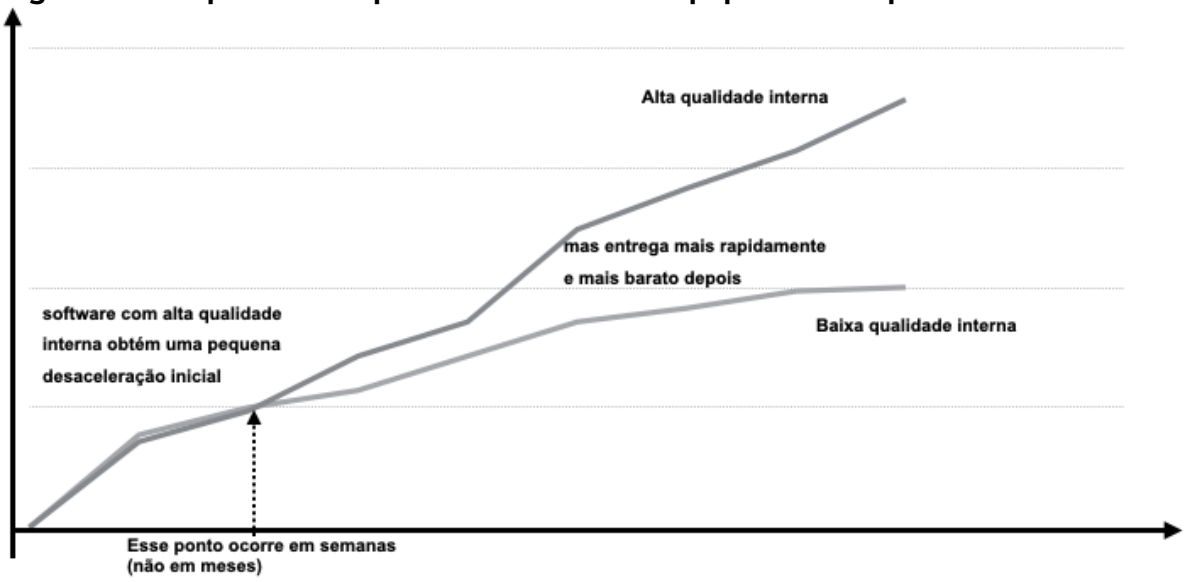
Figura 2 - Gráfico de custo despendido pelas empresas no mesmo cenário da figura 1



Fonte: Adaptado de (MARTIN, 2019).

Na figura 3, encontramos comparativo de produtividade entre projetos com arquiteturas boas e ruins, trazendo como conclusão que a fase de planejamento e desenho de arquitetura de sistemas como uma das fases mais importantes para o projeto, dando a aplicação, maior flexibilidade e aumentando a produtividade das equipes. Há também a tradução do que o autor escreve localizado na figura.

Figura 3 - Comparativo de produtividade entre equipes com arquiteturas boas e ruins



Fonte: Adaptado de (FOWLER, 2020)

1.1 ESTRATÉGIAS DE MODULARIZAÇÃO

Considerando empresas atuais com grandes produtos e que não construíram uma aplicação de forma bem arquitetada, é possível questionar como transformar essas realidades numa nova proposta? Quais são os padrões e estratégias de mercado a utilizar? A aplicação da estratégia de estrangulamento de sistemas é uma possível solução (RICHARDSON, 2021).

Esta estratégia consiste, em encapsular o código atual já produzido em um módulo, isolando-o dentro da aplicação principal, a fim de isolar todo código já construído, terminando o ciclo antigo da aplicação, reduzindo esse grande bloco de funcionalidades em grupos menores de sistemas que agora corresponde ao padrão de implementação adotado pela equipe, removendo as funcionalidades de dentro do monolito até que o legado se dissipe (OROSZ, 2021).

A questão que sempre nos concerne se tratado que fazer com os códigos compartilhados, no caso, as camadas de serviços que utilizam bibliotecas para executar as chamadas ao servidor por exemplo, ou até mesmo as camadas de persistência, navegação e gerenciamento de estados. Encontramos então dentro das funcionalidades a dependência de camadas semelhantes, que são classificados como de módulos horizontais (MARTIN, 2019).

Esses módulos são similares a uma caixa de ferramentas para as equipes de desenvolvimento. Temos então como prioridade construir esses módulos a fim de dar às equipes a capacidade de entregar produtos cada vez mais rápido, visto que as equipes irão sempre se dedicar em entregar módulos focados em atender a regra de negócio e influenciando até mesmo no "toil" das equipes. Nasce então a necessidade de possuir um gerenciador de dependências a fim de orquestrar as dependências que serão injetadas para cada fluxo.

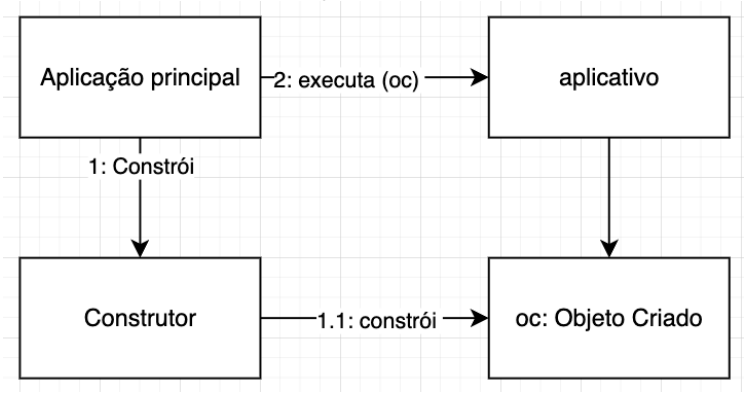
1.2 SEPARAÇÃO DA CAMADA PRINCIPAL

É fundamental que a aplicação tenha na camada principal todos os aspectos de construção ou sistemas menores que sejam chamados pela inicialização do sistema e modelar o resto da aplicação, assumindo que todos os objetos foram atribuídos de forma adequada (MARTIN, 2019).

A figura 4 exemplifica o modelo esperado da inicialização de uma aplicação modularizada. Para o bom funcionamento de aplicação, esta necessita do gerenciamento de suas funcionalidades de forma autônoma, sem depender de módulo de níveis internos, isto é uma

aplicação conhece onde estão os recursos capazes de oferecer as funcionalidades os recursos que habilitam a capacidade de cumprir o propósito para que o módulo foi criado.

Figura 4 - Exemplo de Separação da camada principal da aplicação



Fonte: Martin, 2019

1.3 INJEÇÃO DE DEPENDÊNCIAS

Injeção de dependências em sua forma mais sintetizada é a abordagem utilizada para gerenciar as aplicações modulares do aplicativo (SEEMAN, 2019). Com o objetivo de dar a aplicação principal o controle de navegação, estados e conhecimento das dependências no projeto, tal assunto se coloca como um dos principais desafios de implementar e uma das etapas mais necessárias de serem feitas antes de definitivamente a etapa de construção dos módulos de funcionalidades (OROSZ, 2021). A injeção de dependências pode ser gerenciada de três maneiras: por meio de construtores, interfaces ou propriedades (SEEMAN, 2019). Diferente do Android que utiliza injetores de dependências maduros que analisam o tempo de compilação das dependências, os sistemas iOS não possuem uma biblioteca que garante esse tipo de gerenciamento apartado (OROSZ, 2021).

Utilizar esta técnica para injetar os módulos dão as aplicações, melhor legibilidade sobre cada comportamento da aplicação, aumenta a capacidade de testar as aplicações e contribui para que tópicos como gerenciamento de estados e governança de módulos sejam menos complexos de serem construídos nas aplicações (OROSZ, 2021). Num contexto de módulos, a camada responsável por saber as dependências das classes, é a camada principal do projeto. Assim, a camada principal do projeto teria uma classe baseada no princípio “Root Composition” de injeção de dependências, agindo como um orquestrador do sistema, dando clareza e controle das dependências entre módulos e bibliotecas externas (SEEMAN,2019).

1.4 PADRÕES “GUI”

Padrões “GUI” representam os padrões utilizados nos sistemas para a camada de interface gráfica, sua diferenciação está no fluxo de dados que podem ser classificados entre padrões bidirecionais e padrões unidirecionais (EIDHOF, GALLAGHER, KUGLER, 2018). Os padrões se diferem nas camadas de fluxos de dados (EIDHOF, GALLAGHER, KUGLER, 2018). O objetivo desses padrões em todos os casos é de separar da camada de *design* ou exibição da responsabilidade de gerenciamento de dados, do gerenciamento do sistema operacional, da navegação, e de todas as dependências implícitas, além de outros pontos que necessitam ser solucionados (MARTIN, 2019).

1.4.1 Padrões bidirecionais

Os padrões bidirecionais consistem em receber uma ação do usuário através da camada “View” que se refere a interface visual, como gestos e cliques em componentes da tela e que por sua vez, acionam a camada “presenter” que se refere a regra de apresentação do projeto. O “presenter” é responsável por tratar os insumos provisionados pelo usuário na camada “view” e acionar a camada “interactor” que se refere a interação da aplicação com o caso de uso mapeado nos requisitos da aplicação. É de sua responsabilidade implementar as atividades concernentes ao caso de uso, acessando a camada mais interna do projeto as “models”, referente aos modelos de dados do sistema. Em seguida a camada “interactor”, retorna ao “presenter” o resultado da operação chamada por ele, que por sua vez se responsabiliza por também tratar a resposta da camada anterior e informar a camada visual que a ação foi executada. Dentro do ecossistema Apple, os padrões mais difundidos são o MVC e o MVVM (APPLE, 2018). Existem outros padrões “GUI” bidirecionais, tais como VIPER e RIBs (OROSZ, 2021).

1.4.2 Padrões unidirecionais

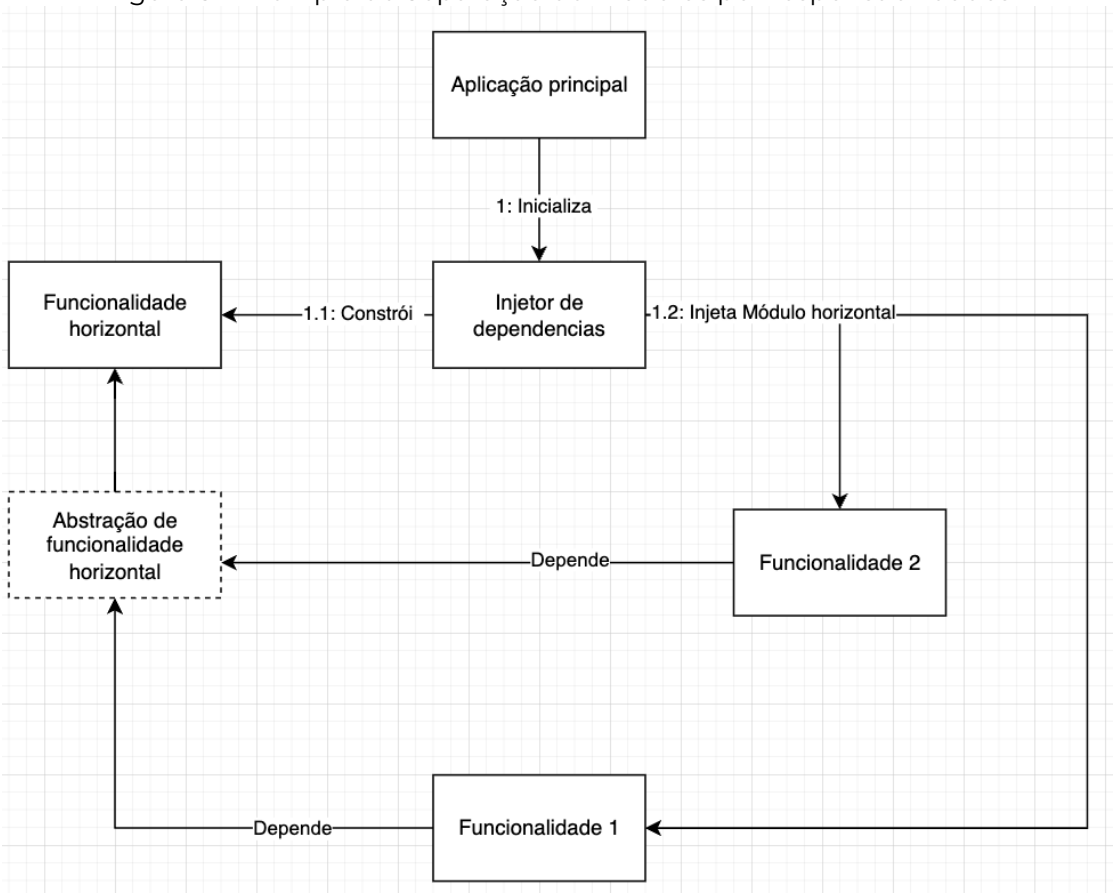
Os padrões unidirecionais assim como os padrões bidirecionais, interagem com o usuário utilizando a mesma camada visual, a “view”, sua diferença mais clara consiste em como ela aciona suas camadas internas, visto que sua camada “view” aciona diretamente a camada de casos de uso “interactor”, que por sua vez solicita a formatação dos resultados obtidos para

que por sua vez, acione a camada “view” com o resultado da operação disparada por ele mesmo. No ecossistema Apple, encontramos os padrões VIP, Clean Swift, como padrões unidirecionais (Clean Swift, 2022).

1.5 O QUE ESPERAR DE UMA APLICAÇÃO MODULAR

A figura 5 a seguir apresenta o diagrama que faz usodearquitetura modular. Suas características principais são, módulos bem delimitados com responsabilidades claras e independentes, onde os times responsáveis por esses módulos atuam para manter, evoluir e corrigir possíveis incidentes e ou emergências.

Figura 5 - Exemplo de Separação de módulos por responsabilidades



Fonte: Elaborado pelo autor

A aplicação apresenta sistema flexível para padrões “GUI” coexistirem de acordo com a necessidade do projeto (ANDROID DEVELOPER, 2022). A aplicação possui um tempo de entrega muito menor, dando aos times mais velocidade em experimentar e testar

funcionalidades, permitindo ligar e desligar módulos em tempo real se gerenciado por ferramentas orientadas por um servidor (FOWLER, 2017).

2. MATERIAL E MÉTODOS

O método proposto para este trabalho se enquadra em uma pesquisa bibliográfica exploratória, pois seu processo de pesquisa tem como objetivo proporcionar maior familiaridade com este assunto e trazer maior detalhamento ou construir hipóteses. A maioria das pesquisas envolve:

a) Levantamento bibliográfico;
b) Análise de exemplos que estimulem a compreensão (GIL, 2022).

As entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado, apesar de fazer parte deste tipo de pesquisa, não será aplicado a proposta deste artigo. Para a formação das conclusões foi utilizado um método de caráter dedutivo, pois parte-se das observações gerais e direciona para o caso particular da investigação.

2.1 Procedimentos técnicos

Em terminologia voltada aos procedimentos técnicos, utilizou-se da pesquisa bibliográfica com objetivo exploratório, através de buscas realizadas em artigos que aplicassem estudos de casos relevantes para o tema abordado, fazendo uso das plataformas bibliográficas “Google Scholar”. Pesquisa realizada no dia 5 de dezembro de 2022. Seus descritores foram respectivamente: {"mobile development"; "iOS"; "mobile Architecture"; “modules”}, estes descritores foram adotados após a realização de testes com outros termos com a finalidade da obtenção de artigos com maior pertinência para o trabalho em questão. As opções de filtro selecionadas no mecanismo de busca foram:

- Período de 2019 até 2022;
- Ordenação por relevância;
- Em qualquer idioma;

- Qualquer tipo
- Incluir citações

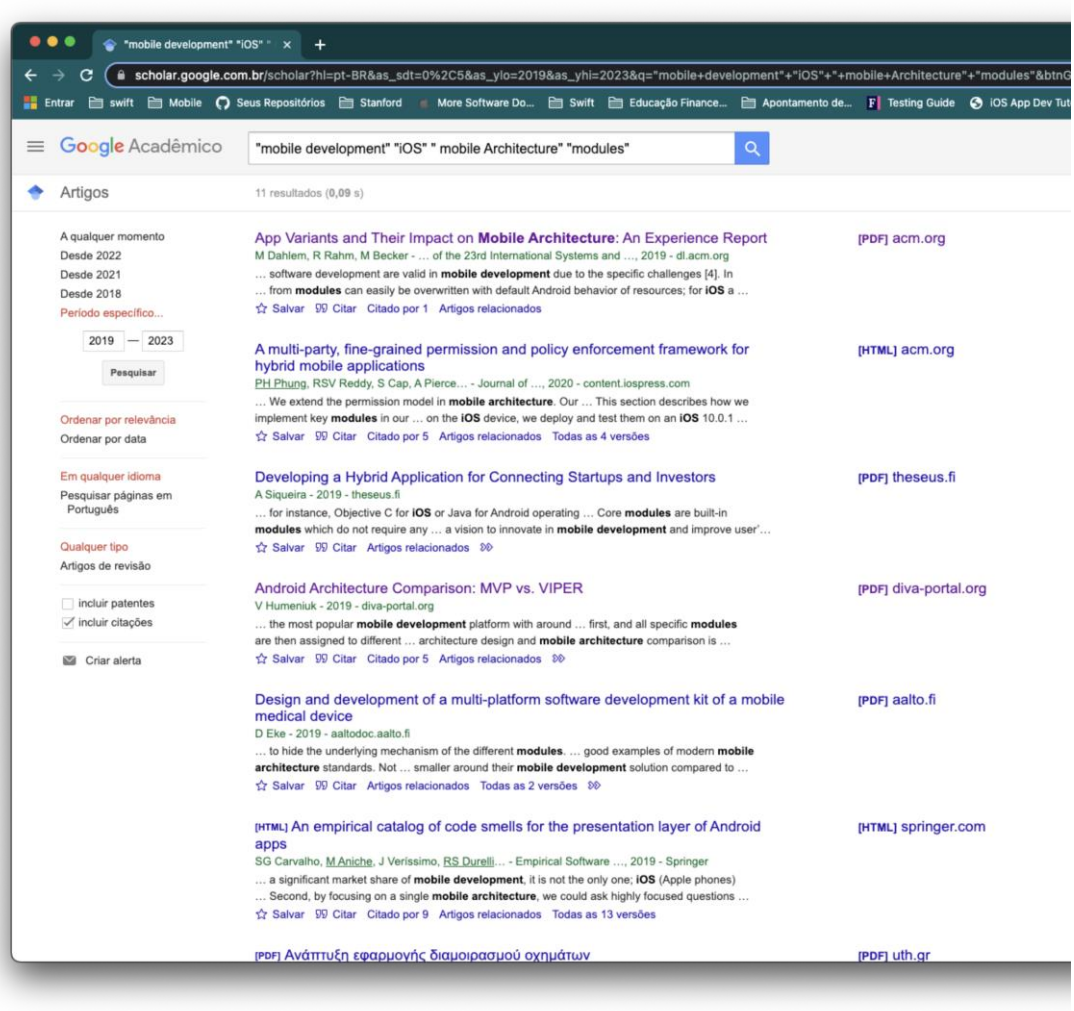
Para segregação do material, ficaram definidas as seguintes regras neste trabalho:

- Alta relevância referente ao tema: documento aborda todos os assuntos descritos no tema (Arquitetura modular para aplicativos iOS, Estratégias de modularização, Injeção de dependências, Padrões GUI).
- Média relevância referente ao tema: documento aborda sistemas monolíticos em plataformas diferentes como Android.
- Baixa relevância referente ao tema: documento não apresenta nenhuma pertinência à pesquisa em questão.

A eleição dos materiais pertinentes foi obtida por meio da leitura exploratória e seletiva no material coletado inicialmente para uma segregação primária. Posteriormente a técnica de leitura analítica e interpretativa. A leitura analítica que visa elencar as informações apresentadas na fonte a fim de possibilitar a obtenção de respostas da pesquisa. Por fim, procede a leitura interpretativa, muitas vezes executada em conjunto com a leitura analítica que visa conectar o conteúdo na fonte com outros conhecimentos (GIL,2022).

Na figura 6, é evidenciado método de busca na plataforma bibliográfica.

Figura 6 – Evidência de pesquisa na plataforma Google Scholar



Fonte: Elaborado pelo autor

3. DISCUSSÃO

O processo de pesquisa na abordagem de Dahlem; Rahm; Becker, (2019) apresenta pontos de extrema relevância para o assunto, contribuindo tecnicamente em estratégias para estabelecer processos de melhoria de produtividade no processo de desenvolvimento, proporcionando etapas importantes para alcançar uma arquitetura de sistema modular, entretanto se faz necessário maior detalhamento no que tange os temas de separação de camadas da aplicação principal do projeto, e a utilização de injeção de dependências.

4 CONSIDERAÇÕES FINAIS

Construir uma aplicação modular implica em analisar diversos conceitos de engenharia de software, dominar diversos padrões de arquitetura e muita experimentação. Sabemos que reestruturar produtos que já estão publicados, submetidos ao uso do cliente final, traz um grande desafio ao desenvolvedor, mostrar valor a companhia na evolução da plataforma além de melhorias das funcionalidades. Esse propósito é bastante difícil de ser sustentado nas companhias e esse é um dos maiores desafios das equipes que necessitam fazer essa transformação digital.

De fato, uma aplicação modular oferece diversos benefícios às equipes que buscam adotar de forma correta, respeitando os padrões de arquitetura, evitando pular etapas, e prezando por uma régua alta de qualidade, onde os testes de unidade, testes integrados e até mesmo os testes automatizados se completam e garantem a qualidade (DAHLEM; RAHM; BECKER, 2019).

Em contrapartida, temos alguns pontos importantes que entram como desafios culturais no processo de oferecer à companhia uma aplicação modular. Encontramos uma resistência grande ao novo e diferente, a dificuldade em disseminar o conhecimento, o esforço em estimular a evolução do time rumo a modularização de um aplicativo que embora demore para entregar as funcionalidades, está funcionando.

A resistência ao novo, a curva de aprendizado, a pequena quantidade de literaturas que sintetize os assuntos e desafios da arquitetura modular e até mesmo temas como topologia de times tendem a ser desafios esperados quando entramos no tema "modularização de plataforma". Além destes itens, os problemas como, gerenciamento de esteira de integração contínua, armazenamento de código fonte com múltiplos repositórios ou de repositório único são discussões longas e devem ser tomadas em conjunto com as equipes, acompanhadas e medidas a fim de corrigir as decisões que em algum momento podem deixar de fazer sentido durante a construção dessa nova fase de projeto.

Mesmo entendendo que modularizar em casos de grandes equipes com aplicações com jornadas grandes é preciso, devemos considerar as dores que enfrentaremos para um fim maior, entendendo que o propósito de entregar esse tipo de estratégia é evoluir a plataforma com mais velocidade, dar aos times de produtos maior facilidade em experimentar mais estratégias, em menos tempo, com flexibilidade e segurança de que suas alterações não impactarão outras jornadas.

As etapas técnicas de desbravamento do código, entender abordagem do código anterior, mapear níveis de acoplamento de código, dependências implícitas em cada funcionalidade, gerenciamento de estados e navegação, são desafios grandiosos para a equipe de engenharia.

Ao utilizar os padrões e estratégias de estrangulamento de sistemas (RICHARDSON, 2021), conseguimos fechar o ciclo de desenvolvimento acoplado e por isso, se faz necessário o isolamento do código atual enquanto criam novos módulos que irão servir as novas funcionalidades e que irão substituir as que foram construídas anteriormente, damos a aplicação a liberdade de escolher o padrão e arquitetura mais indicado para cada funcionalidade, obtendo por consequência um projeto construído de forma modular, isso garante que as tecnologias e estratégias servirão ao propósito, assim não impomos a aplicação a uma definição imperativa e imutável (OROSZ,2021). Fazer uso desse modelo de desenvolvimento de aplicativos contribui para que cada equipe apresente um propósito bem definido, aumenta a produtividade dos colaboradores habilita as companhias sistemas mais saudáveis e flexíveis.

REFERÊNCIAS

ANDROID DEVELOPER. **Guia para arquitetura do app**. Disponível em: https://developer.android.com/topic/architecture. US: Android Developer, 2022.

CLEAN SWIFT. **Manual do Clean Swift**. Disponível em: https://clean-swift.com/handbook/. US: Clean Swift LLC, 2022.

DAHLEM, Marc; RAHM, Ricarda; BECKER, Martin. **App Variants and Their Impact on Mobile Architecture: An Experience Report**. Disponível em: https://dl.acm.org/doi/abs/10.1145/3336294.3336320. US: Association for Computer Machinery, 2019.

DEVELOPER APPLE. MODEL VIEW CONTROLLER. Disponível em: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html. US: Developer Apple, 2018.

EIDHOF, Chris; GALLAGHER, Matt; KUGLER, Florian. **App Architecture: iOS** Application Patterns in Swift. US: objc,2018.

EVANS, Eric. **Domain-Driven Design: Atacando as complexidades no coração do Software**. [S.I.] Alta Books 2017.

FOWLER, Martin. **Refatoração: Aperfeiçoando o Design de Códigos Existentes**. [S. l.] Novatec, 2020.

FOWLER, J Susan. **Microsserviços prontos para a produção: Construindo sistemas padronizados em uma organização de engenharia de software**. [S.I.], Novatec, 2017.

GARTNER. **GUI (Graphical User Interface).** Disponível em: https://www.gartner.com/en/information-technology/glossary/gui-graphical-user-interface. US: Gartner, 2022.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. 7. ed. São Paulo, Atlas, 2022.

GOOGLE CLOUD**. Eliminating Toil**. Disponível em: https://sre.google/sre-book/eliminating-toil/. US: SRE Google, 2017.

MARTIN, Robert C. **Arquitetura limpa: O guia do artesão para estrutura e design de software**. [S. l.], Alta Books 2019.

MICROSOFT. **Containerized Docker Application Lifecycle with Microsoft Platform and Tools**. US: Microsoft, 2022.

OROSZ, Gergely. **Building Mobile Apps at Scale**. Disponível em: https://www.mobileatscale.com/. US: Mobile At Scale, 2021.

RICHARDSON, Chris. **Aplicação Strangler**. Disponível em: https://microservices.io/patterns/refactoring/strangler-application.html. Microservices.io, 2021

SEEMAN, Mark. **Dependency Injection Principles, Practices, and Patterns**. US: Manning Publications, 2019.

VERNON, Vaughn. **Implementando Domain-Driven Design**. [S.I.], Alta Books 2016.