UNIVERSIDADE FEDERAL DO CEARÁ

INSTITUTO UFC VIRTUAL

BACHARELADO EM SISTEMAS E MÍDIAS DIGITAIS

ANDERSON DE ALENCAR BEZERRA SOUZA

ANÁLISE COMPARATIVA DE ARQUITETURAS MOBILE PARA APLICAÇÕES IOS: MVC, MVVM E VIP-C

FORTALEZA - CE

2026

ANDERSON DE ALENCAR BEZERRA SOUZA

ANÁLISE COMPARATIVA DE ARQUITETURAS MOBILE PARA APLICAÇÕES IOS: MVC, MVVM E VIP-C

Monografia apresentada ao Curso de Sistemas e Mídias Digitais da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Sistemas e Mídias Digitais.

Orientadora: Profa. Dra. Marília Soares Mendes Albuquerque.

FORTALEZA - CE

2026

ANDERSON DE ALENCAR BEZERRA SOUZA

ANÁLISE COMPARATIVA DE ARQUITETURAS MOBILE PARA APLICAÇÕES IOS: MVC, MVVM E VIP-C

Monografia apresentada ao Curso de Sistemas e Mídias Digitais da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Sistemas e Mídias Digitais.

Aprovada em: 29 de janeiro de 2026

BANCA EXAMINADORA

_____
Profa. Dra. Marília Soares Mendes Albuquerque (Orientadora)
Universidade Federal do Ceará (UFC)

_____
Profa. Dra. Amanda Drielly Pires Venceslau
Universidade Federal do Ceará (UFC)

_____
Prof.Dr. José Gilvan Rodrigues Maia
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

A realização deste trabalho foi possível graças ao apoio e à contribuição de pessoas e instituições que marcaram de forma inimaginável minha trajetória acadêmica.

Agradeço à minha orientadora, Prof.ª Dra. Marília Soares Mendes Albuquerque, pelo rigor acadêmico, pela paciência, pela resiliência e pelas contribuições fundamentais ao longo de todo o processo de desenvolvimento deste trabalho.

À Universidade Federal do Ceará (UFC), expresso minha gratidão pela formação acadêmica e pelas oportunidades que possibilitaram a construção de uma trajetória profissional pautada na dedicação, no esforço e na realização de objetivos pessoais e acadêmicos.

Agradeço profundamente ao meu marido, Marcos José, pelo apoio constante, pela presença sempre inspiradora e pelo incentivo em todas as fases da graduação, que me deram força e confiança para essa conquista.

Aos meus amigos, agradeço pelo apoio, pela participação no processo de coleta de dados deste trabalho e pelos momentos de inspiração e equilíbrio que contribuíram para a realização desta pesquisa.

À minha família, em especial à minha mãe, Antônia Danielle, e ao meu pai, José de Alencar, agradeço pelo amor, pelo apoio incondicional e pelos ensinamentos que foram fundamentais para minha formação pessoal e acadêmica.

Por fim, e de forma muito especial, agradeço à Charlotte, minha eterna companheira. Presente em dias e noites de estudo, participante silenciosa e inabalável de inúmeros projetos e trabalhos ao longo da minha graduação, esteve comigo nos momentos mais desafiadores e também nos mais solitários. Sua companhia foi conforto, sua presença foi luz. Dedico a ela este trabalho, com a certeza de que, onde quer que esteja, permanece parte essencial de tudo o que sou e de tudo o que conquistei.

“O real não está na saída nem na chegada: ele se dispõe para a gente é no meio da travessia.” *(ROSA, 1956, p. 52).*

RESUMO

O planejamento do desenvolvimento de aplicações mobile no ecossistema iOS envolve decisões arquiteturais que podem impactar diretamente a organização do código, a evolução do sistema e a qualidade do software ao longo do tempo. Dentre essas decisões, a escolha da arquitetura de software do sistema assume papel central, devendo levar em consideração os requisitos técnicos, os objetivos de negócio e as intenções de crescimento do projeto. Nesse cenário, surge a seguinte questão: de que forma as arquiteturas MVC, MVVM e VIP-C influenciam atributos de qualidade de software e em quais contextos de projeto cada uma se mostra mais adequada no desenvolvimento de aplicações iOS. A diversidade de abordagens adotadas na prática reforçam a relevância dessa investigação. O objetivo deste trabalho é realizar uma análise comparativa entre as arquiteturas *Model–View–Controller* (MVC), *Model–View–ViewModel* (MVVM) e *View–Interactor–Presenter–Coordinator* VIP-C, identificando seus impactos sobre atributos de qualidade, como modularização, acoplamento, coesão, fluxo de dados, testabilidade e manutenibilidade, bem como estabelecer diretrizes que auxiliem na escolha arquitetural para diferentes cenários de projeto no contexto iOS. Para atingir esse objetivo, foi adotada uma metodologia qualitativa e analítica, baseada em revisão da literatura científica e técnica, envolvendo estudos empíricos, conceituais e aplicados. Os trabalhos selecionados foram analisados de forma comparativa, considerando a distribuição de responsabilidades entre componentes, os mecanismos de comunicação entre camadas. Os resultados indicaram que o MVC é predominantemente adotado em aplicações iniciais e de menor complexidade, devido à sua simplicidade estrutural e integração com os frameworks nativos do iOS; que o MVVM se mostra mais adequado a projetos de pequeno e médio porte, especialmente no contexto do SwiftUI, ao favorecer a separação entre lógica de apresentação e interface; e que arquiteturas mais segmentadas, como o VIP-C, apresentam melhor desempenho em termos de modularização, testabilidade e manutenibilidade em sistemas de médio e grande porte, ao custo de maior esforço inicial de implementação.

Palavras-chave: Arquitetura de Software; Desenvolvimento iOS; MVC; MVVM; VIP-C.

ABSTRACT

Planning the development of mobile applications in the iOS ecosystem involves architectural decisions that can directly impact code organization, system evolution, and software quality over time. Among these decisions, the choice of software architecture plays a central role and must consider technical requirements, business objectives, and the intended growth of the project. In this context, the following research question arises: how do the MVC, MVVM, and VIP-C architectures influence software quality attributes, and in which project contexts is each architecture more suitable for iOS application development? The diversity of approaches adopted in practice reinforces the relevance of this investigation. The objective of this study is to conduct a comparative analysis of the Model–View–Controller (MVC), Model–View–ViewModel (MVVM), and View–Interactor–Presenter–Coordinator (VIP-C) architectures, identifying their impacts on quality attributes such as modularization, coupling, cohesion, data flow, testability, and maintainability, as well as to establish guidelines to support architectural decision-making across different project scenarios in the iOS context. To achieve this objective, a qualitative and analytical methodology was adopted, based on a review of scientific and technical literature, including empirical, conceptual, and applied studies. The selected works were analyzed comparatively, considering the distribution of responsibilities among components and the communication mechanisms between architectural layers. The results indicate that MVC is predominantly adopted in initial and low-complexity applications due to its structural simplicity and integration with native iOS frameworks; that MVVM is more suitable for small- to medium-sized projects, especially in the SwiftUI context, as it promotes a clearer separation between presentation logic and user interface; and that more segmented architectures, such as VIP-C, demonstrate superior performance in terms of modularization, testability, and maintainability in medium- and large-scale systems, at the cost of greater initial implementation effort.

Keywords: Software Architecture; iOS Development; MVC; MVVM; VIP-C.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
IBGE	Instituto Brasileiro de Geografia e Estatística
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Delivery
CPU	Central Processing Unit
IBGE	dfdfdf
iOS	iPhone Operating System
MVC	Model–View–Controller
MVVM	Model–View–ViewModel
MVP	Model–View–Presenter
MVI	Model–View–Intent
QP	Questão de Pesquisa
RSL	Revisão Sistemática da Literatura
RxSwift	Reactive Extensions for Swift
SAAM	Software Architecture Analysis Method
SMD	Sistemas e Mídias Digitais
SwiftUI	Framework declarativo da Apple para desenvolvimento de interfaces
UFES	Universidade Federal do Espírito Santo
UFC	Universidade Federal do Ceará
UIKit	Framework da Apple para construção de interfaces gráficas em iOS
VIP	View–Interactor–Presenter
VIP-C	View–Interactor–Presenter–Coordinator
VIPER	View–Interactor–Presenter–Entity–Router
WPF	Windows Presentation Foundation

SUMÁRIO

1 INTRODUÇÃO

Segundo Bass, Clements e Kazman (2013), a arquitetura de software é “a estrutura (ou estruturas) do sistema, composta de elementos de software, das propriedades externamente visíveis desses elementos e dos relacionamentos entre eles” (BASS; CLEMENTS; KAZMAN, 2013, p. 15). Essa definição evidencia que a arquitetura não se limita à implementação do código, mas estabelece um modelo estrutural de alto nível que orienta como os componentes de um sistema são organizados e como interagem entre si. Nesse sentido, a arquitetura exerce papel central na condução das decisões técnicas, influenciando diretamente atributos de qualidade como modularização, manutenibilidade, testabilidade e capacidade de evolução do software. Ao fornecer uma visão global do sistema, a arquitetura atua como base para o planejamento, o desenvolvimento e a manutenção de aplicações, tornando-se elemento essencial para a construção de soluções robustas e sustentáveis ao longo do tempo.

Nas últimas duas décadas, a expansão acelerada das tecnologias móveis alterou profundamente a forma como aplicações são projetadas, desenvolvidas e mantidas. Smartphones deixaram de ser ferramentas acessórias e se tornaram plataformas que centralizam comunicação, consumo de conteúdo, gestão financeira, educação e inúmeros outros serviços digitais. Segundo relatório da Data.ai (2023), usuários de dispositivos móveis passam, em média, 5 horas por dia interagindo com aplicativos, e o mercado global de mobile ultrapassou a marca de US$ 500 bilhões movimentados apenas em 2022 Data.ai (2023). No ecossistema iOS, a Apple (2023) reportou que mais de 90% dos aparelhos ativos rodam versões recentes do sistema operacional, pressionando desenvolvedores e empresas a manterem ciclos de atualização curtos, produtos estáveis e arquiteturas que suportem evolução constante.

Todo esse cenário coloca em foco o elemento chave da evolução de softwares especializados, a Arquitetura de Software, elemento essencial para orientar a organização interna de sistemas e garantir que aplicações cresçam sem se tornarem rígidas ou frágeis. A literatura clássica demonstra que até 70% do custo de vida de um sistema está relacionado à manutenção (Pressman, 2020). Em aplicações mobile, esse desafio torna-se especialmente relevante devido à necessidade de rápida adaptação a novas APIs, requisitos de design,

atualizações de segurança e mudanças nas diretrizes das lojas de aplicativos. Uma arquitetura que privilegia modularização, testabilidade e clareza estrutural se torna, portanto, um diferencial estratégico.

No contexto do desenvolvimento para o ecossistema iOS, esse desafio assume características específicas. O ecossistema Apple evoluiu rapidamente com a transição da linguagem de programação orientada a objetos Objective-C para a linguagem Swift, lançada em 2014, a qual introduziu novos conceitos de organização de código, segurança de tipos e modularização, impactando diretamente a forma como aplicações são estruturadas (Apple INC., 2014). Além disso, a introdução do framework SwiftUI em 2019 representou uma mudança de paradigma no desenvolvimento de interfaces, ao adotar um modelo declarativo baseado em estado, no qual a interface passa a refletir diretamente os dados da aplicação (Apple INC., 2019). Essas transformações ampliaram a necessidade de arquiteturas de software mais bem definidas, capazes de organizar o fluxo de dados, separar responsabilidades e sustentar a evolução contínua das aplicações iOS.

De acordo com o Stack Overflow Developer Survey (2023), a Swift está entre as linguagens mais bem avaliadas, em termos de satisfação, por desenvolvedores, mas também entre as que apresentam maior curva de aprendizado inicial, exigindo redução de complexidade, para que a plataforma seja amigável para antigos e novos desenvolvedores.

Ao mesmo tempo, diferentes estudos na área de Engenharia de Software destacam que organizações contemporâneas têm buscado arquiteturas de software capazes de sustentar equipes ágeis, favorecer escalabilidade e facilitar a adoção de testes automatizados. Neto (2025) argumenta que a arquitetura exerce papel central na manutenção da qualidade do software em contextos ágeis, especialmente em sistemas de médio e grande porte, nos quais a modularização e a separação de responsabilidades são fundamentais para a adaptação contínua do sistema. De forma complementar, Shahin e Babar (2020) demonstram, a partir de estudos industriais, que arquiteturas modulares contribuem para a adoção de práticas como testes automatizados e integração contínua, ao reduzir o acoplamento entre componentes e facilitar a evolução incremental do código.

Nesse contexto, três arquiteturas se destacam de forma consistente no ecossistema iOS, tanto em materiais de formação quanto na literatura técnica e no mercado profissional. A

plataforma educacional Cursa, referência nacional em cursos de desenvolvimento mobile, classifica as arquiteturas MVC, MVVM e VIPER/VIP-C como “arquiteturas fundamentais no iOS” (Cursa, 2025), apresentando-as como estrutura base para qualquer desenvolvedor que pretende atuar na área. No campo acadêmico, o estudo de Gomes (2023) identifica essas mesmas arquiteturas como padrões abundantemente difundidos em aplicações móveis, destacando a MVC pela simplicidade inicial, a MVVM pela separação clara de responsabilidades e escalabilidade, e a VIPER/VIP-C pelo elevado grau de modularização. Essa relevância também se confirma na prática profissional: vagas de desenvolvedor iOS de empresas nacionais como Nubank frequentemente listam MVC, MVVM ou VIPER/VIP-C como requisitos desejáveis (Nubank, 2025), evidenciando sua adoção consolidada no setor.

Embora existam múltiplas arquiteturas disponíveis para o desenvolvimento iOS, a prática profissional demonstra que muitos projetos só passam a adotar padrões arquiteturais mais estruturados quando problemas de manutenção e crescimento começam a se tornar críticos (Clean Swift, 2019). Um dos indícios mais recorrentes desse cenário é o surgimento dos chamados *Massive View Controllers*, termo utilizado na comunidade iOS para descrever controladores de interface que concentram excessivamente responsabilidades, incluindo lógica de apresentação, regras de negócio e coordenação de fluxo, comprometendo a modularização e a manutenibilidade do código (Law, 2015; Clean Swift, 2019). Nesse antipadrão, o controlador acumula lógica de apresentação, regras de negócio, manipulação de dados e controle de navegação, resultando em código extenso, fortemente acoplado e de difícil manutenção (Law, 2015; Clean Swift, 2019).

Esse fenômeno é documentado em materiais técnicos especializados. A documentação do Clean Swift (2019) apresenta exemplos reais de controladores que se tornam inchados e difíceis de manter quando não existe uma separação adequada de responsabilidades, evidenciando como a ausência de uma arquitetura bem definida compromete a estabilidade, a modularidade e a evolução do código. De forma semelhante, Code With Shabib (2022) destaca que o *Massive View Controller* surge com frequência em projetos baseados em um uso intuitivo e não planejado do MVC. Relatos de migração arquitetural, como os publicados pela empresa Eventogy Inc., documentam a transição de MVC para MVVM em seus projetos iOS, descrevendo problemas como *Massive View Controllers*, dificuldade de realizar testes unitários e acoplamento excessivo, que motivaram a mudança para uma arquitetura mais modular, com melhoria da manutenção e da testabilidade (Halai, 2024).

Além do problema dos *Massive View Controllers*, a literatura e os materiais técnicos analisados discutiram outros desafios recorrentes no desenvolvimento de aplicações iOS quando a arquitetura não é adequadamente planejada. Estudos identificaram dificuldades associadas à baixa testabilidade, à elevada dependência entre componentes e à ausência de delimitação clara de responsabilidades entre camadas. Tais aspectos são apontados na literatura como fatores que dificultam a evolução incremental do sistema e a adoção de testes automatizados (Sholichin et al., 2019; Dobrean; Diosan, 2019). Também foram descritos problemas na organização do fluxo de dados e de navegação, especialmente em aplicações com múltiplos fluxos e estados, onde a lógica de controle tende a se dispersar por diferentes controladores, o que é discutido como um fator de redução da previsibilidade estrutural e de aumento da complexidade de manutenção (Dobrean, 2019). De forma geral, esses problemas foram tratados como elementos que afetam negativamente atributos internos de qualidade, como modularização, manutenibilidade e testabilidade, reforçando a necessidade de uma análise criteriosa das arquiteturas adotadas no contexto iOS.

Diante desse contexto, torna-se evidente que a escolha da arquitetura de software não é uma decisão neutra, é um desafio que influencia diretamente a forma como uma aplicação é estruturada, evolui e é mantida ao longo do tempo. No desenvolvimento mobile para o ecossistema iOS, essa decisão assume relevância especial diante da diversidade de cenários possíveis, que vão desde provas de conceito e projetos com escopo ainda indefinido até aplicações de médio e grande porte que demandam maior robustez técnica e capacidade de evolução. Nesse cenário, coloca-se a seguinte questão de pesquisa: de que forma diferentes arquiteturas mobile aplicadas ao desenvolvimento iOS influenciam aspectos como organização estrutural, modularização e manutenibilidade das aplicações, considerando diferentes contextos e necessidades de projeto? A investigação dessa questão motiva o presente trabalho.

Com base nesta problemática, este trabalho tem como objetivo realizar uma análise comparativa entre arquiteturas de software utilizadas no desenvolvimento de aplicações para o ecossistema iOS, discutindo como as características de cada abordagem podem se mostrar mais ou menos adequadas conforme o contexto e as necessidades do projeto. A investigação concentra-se nas arquiteturas MVC, MVVM e VIP-C, selecionadas por sua relevância histórica, adoção prática e recorrência tanto na literatura técnica quanto no mercado profissional. Assim, busca-se compreender em quais cenários cada arquitetura tende a ser

mais favorável, considerando desde aplicações em fase inicial com escopo ainda indefinido até sistemas de médio e grande porte que demandam maior organização e capacidade de evolução.

Para viabilizar essa comparação de forma estruturada e coerente, o estudo estabelece um conjunto de critérios comparativos que orientam a análise das arquiteturas selecionadas. Esses critérios são definidos a partir da literatura e das práticas descritas por autores e publicações técnicas do ecossistema iOS, buscando garantir consistência na comparação e alinhamento com os objetivos da pesquisa.

Do ponto de vista metodológico, o trabalho adota uma abordagem baseada em revisão sistemática da literatura e em uma investigação complementar por meio de questionário online aplicado a desenvolvedores iOS atuantes no mercado. A revisão sistemática é utilizada para identificar e analisar estudos e materiais relevantes sobre arquiteturas de software no contexto mobile, enquanto o questionário busca coletar percepções práticas sobre a adoção e os desafios associados às arquiteturas investigadas. A integração dessas duas etapas fundamenta a análise comparativa e apoia a elaboração de diretrizes de recomendação de escolha arquitetural ao final do trabalho.

1.1 Motivação

A motivação para este trabalho parte da experiência do autor ao longo de mais de sete anos atuando como profissional no desenvolvimento de aplicações mobile para iOS.

A delimitação do estudo ao ecossistema iOS justifica-se pela familiaridade profissional do autor com a plataforma, construída ao longo dessa trajetória de atuação prática. Essa vivência possibilita maior profundidade analítica, precisão técnica e segurança na interpretação dos aspectos arquiteturais discutidos ao longo do trabalho. Além disso, os resultados obtidos apresentam aplicabilidade prática imediata no contexto profissional do autor, contribuindo diretamente para sua área de atuação. Dessa forma, a restrição do escopo ao iOS não apenas torna o estudo mais viável e focado, como também reforça sua relevância técnica e acadêmica.

Eu vivenciei diversos cenários recorrentes em empresas de diferentes portes, especialmente relacionados a projetos que nascem pequenos, muitas vezes como provas de

conceito, produtos piloto ou iniciativas experimentais e que, posteriormente, passaram a ser produtos de sucesso rentáveis que precisavam ser escalados. Nessas situações é comum que as equipes evitem reescrever o software do zero ou redefinir sua arquitetura, priorizando a continuidade das entregas em detrimento de um planejamento estrutural adequado. Essa decisão, embora compreensível em contextos corporativos, frequentemente resulta em dificuldades significativas de manutenção, causando aumento do acoplamento entre componentes, baixa testabilidade e entraves para implementação de novas funcionalidades.

Grande parte desses problemas decorre de escolhas arquiteturais feitas nos estágios iniciais do projeto, quando a preocupação dominante costuma ser a velocidade de prototipação, e não a sustentabilidade técnica do produto. Desenvolvedores responsáveis por iniciar essas aplicações, muitas vezes por limitações de experiência ou por ausência de diretrizes internas claras, deixam de considerar aspectos essenciais como modularização, fluxo de dados, separação de responsabilidades e escalabilidade, fatores que, mais tarde, impactam diretamente no esforço necessário para evoluir o código e atender às demandas de negócio.

Essa realidade também se manifesta no contexto acadêmico, especialmente no meu Curso de Sistemas e Mídias Digitais da Universidade Federal do Ceará, onde os estudantes são estimulados a desenvolver produtos em curto tempo (tempo da disciplina) e com alto grau de paralelização, no qual os integrantes executam de forma simultânea múltiplas atividades acadêmicas, projetos e disciplinas ao longo do mesmo semestre, desde os semestres iniciais. Embora essa abordagem favoreça a prática e a experimentação, a insuficiência de formação aprofundada em arquitetura de software faz com que muitos projetos enfrentem gargalos estruturais relevantes. A ausência de padrões arquiteturais, aliada à pressão de prazos e à inexperiência técnica dos alunos, frequentemente resulta em código de difícil manutenção, retrabalho extensivo e dificuldades na conclusão das funcionalidades previstas. Tais problemas, por vezes, são interpretados como falhas de gestão de tempo, quando, na verdade, refletem a complexidade arquitetural não planejada desde o início.

Diante disso, este trabalho também se justifica como uma contribuição prática para estudantes e desenvolvedores em formação, ao oferecer um estudo comparativo que esclarece vantagens, limitações e implicações da escolha entre diferentes arquiteturas no ecossistema iOS. A intenção é fornecer subsídios conceituais e práticos que auxiliem equipes a tomar

decisões mais adequadas em diferentes estágios de seus projetos, reduzindo custos posteriores de manutenção e promovendo maior sustentabilidade técnica ao longo do ciclo de vida do software.

1.2 Objetivo Geral

O presente trabalho tem como objetivo geral apresentar resultados de uma análise comparativa entre as arquiteturas de software: MVC, MVVM e VIP-C, no contexto do desenvolvimento de aplicações para iOS.

1.3 Objetivos Específicos

Como objetivos específicos, este trabalho visa:

1. Apresentar os fundamentos conceituais das arquiteturas MVC, MVVM e VIP-C, destacando seus princípios estruturais e suas características;
2. Selecionar critérios comparativos das arquiteturas, a partir de uma revisão sistemática;
3. Comparar as arquiteturas MVC, MVVM e VIP-C com base nos critérios selecionados;
4. Identificar vantagens e limitações de cada arquitetura no contexto do desenvolvimento mobile para iOS;
5. Realizar uma investigação empírica com profissionais do ecossistema iOS, visando confrontar evidências práticas com os resultados da análise da literatura.
6. Apresentar diretrizes de escolha arquitetural, resultante da comparação entre MVC, MVVM e VIP-C, de modo a auxiliar desenvolvedores a selecionarem a abordagem mais adequada para diferentes cenários de projeto.

1.4 Procedimentos Metodológicos

Esta seção descreve os procedimentos metodológicos adotados para a condução da pesquisa, detalhando o tipo de estudo realizado, as etapas da investigação e as estratégias utilizadas para a análise comparativa das arquiteturas MVC, MVVM e VIP-C no contexto do desenvolvimento de aplicações iOS. A metodologia foi estruturada para garantir rigor científico, coerência com os objetivos propostos e alinhamento com a literatura acadêmica da área de Engenharia de Software

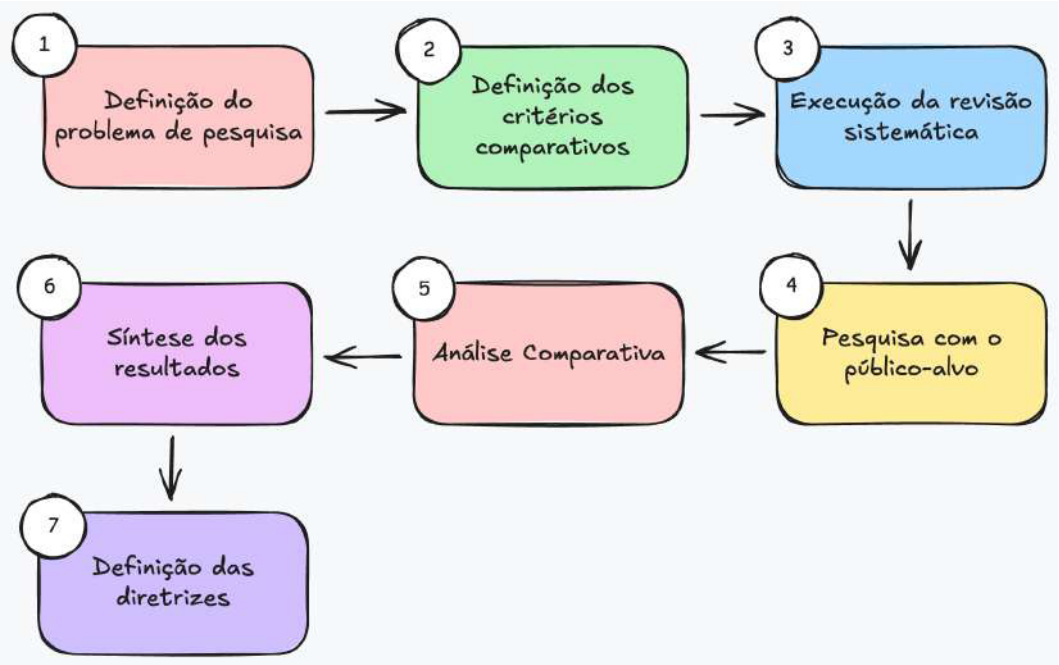
1.4.1 Tipo de pesquisa

A presente pesquisa caracteriza-se como qualitativa, de natureza descritiva e comparativa, com abordagem metodológica baseada em revisão sistemática da literatura e investigação empírica complementar por meio de questionário online aplicado a desenvolvedores iOS.

A abordagem qualitativa é adequada por permitir a análise aprofundada de características estruturais, decisões arquiteturais e implicações práticas associadas às arquiteturas estudadas. O caráter descritivo possibilita a exposição detalhada dos conceitos, enquanto a abordagem comparativa orienta a análise das diferenças e semelhanças entre MVC, MVVM e VIP-C a partir de critérios previamente definidos.

Com o objetivo de organizar e tornar explícitas as etapas adotadas neste estudo, a Figura 1 apresenta uma visão geral da metodologia utilizada, evidenciando a sequência dos procedimentos realizados, desde a definição do problema de pesquisa até a síntese dos resultados e diretrizes de análise.

Figura 1 – Diagrama da metodologia adotada no estudo



Fonte: Elaborado pelo autor (2026).

1.4.2 Definição do problema de pesquisa

Definição do problema de pesquisa, iniciou-se a partir da identificação de lacunas e divergências na literatura e na prática profissional quanto à escolha de arquiteturas de software no desenvolvimento de aplicações mobile para o ecossistema iOS. Observou-se que arquiteturas como MVC, MVVM e VIP-C são utilizadas, porém com diferentes implicações sobre atributos de qualidade e uso distintos em contextos de projeto.

1.4.3 Critérios de comparação

A análise comparativa das arquiteturas MVC, MVVM e VIP-C foi conduzida com base nos seguintes critérios, selecionados com base em referências recorrentes na literatura sobre arquitetura de software e desenvolvimento mobile:

- Modularização: grau de divisão do sistema em componentes independentes;
- Fluxo de dados: modo como informações circulam entre as camadas;
- Acoplamento e coesão: nível de dependência entre módulos e clareza das responsabilidades internas;
- Testabilidade: facilidade de criação e isolamento de testes automatizados;
- Manutenibilidade: esforço necessário para correções, ajustes e evolução do sistema;

Esses critérios orientaram tanto a análise teórica quanto a interpretação dos dados coletados no questionário.

1.4.4 Revisão sistemática da literatura

A revisão sistemática teve como objetivo identificar, selecionar e analisar trabalhos acadêmicos e técnicos relevantes relacionados ao uso de arquiteturas de software no desenvolvimento mobile, com ênfase no ecossistema iOS.

O processo seguiu etapas inspiradas em diretrizes consolidadas da literatura científica, envolvendo:

- definição do problema de pesquisa;
- estabelecimento de critérios de inclusão e exclusão;
- seleção de bases e fontes confiáveis;
- análise crítica dos estudos selecionados.

Foram considerados artigos científicos, trabalhos de conclusão de curso, dissertações, documentação técnica oficial e publicações reconhecidas da comunidade iOS, desde que abordassem arquiteturas como MVC, MVVM, VIPER/VIP-C ou padrões correlatos, bem como atributos de qualidade como modularização, testabilidade, manutenibilidade, fluxo de dados, acoplamento e coesão.

1.4.5 Investigação empírica com desenvolvedores

Como complemento à revisão sistemática, foi conduzida uma investigação empírica por meio de questionário online, direcionado a desenvolvedores com experiência no desenvolvimento de aplicações iOS. O instrumento de coleta foi aplicado por meio da plataforma Google Forms, escolhida por sua acessibilidade, facilidade de uso e suporte à coleta e organização automática das respostas.

O questionário teve caráter exploratório e buscou coletar percepções práticas sobre:

- arquiteturas mais utilizadas no dia a dia profissional;
- critérios considerados na escolha arquitetural;
- dificuldades enfrentadas em projetos reais;
- impacto das arquiteturas na manutenção, testabilidade, organização, modularidade e coesão do código .

Os dados coletados foram analisados de forma qualitativa, servindo como apoio para confrontar a literatura com a prática profissional, sem pretensão de generalização estatística.

1.4.6 Análise comparativa

A partir da integração entre a revisão sistemática e a investigação empírica, foi realizada uma análise comparativa estruturada das arquiteturas estudadas. Os resultados foram organizados, permitindo identificar vantagens, limitações e cenários de aplicação mais adequados para cada abordagem arquitetural.

Essa síntese fundamenta as diretrizes de escolha arquitetural apresentada nos capítulos finais do trabalho.

1.4.7 Delimitações da pesquisa e definição das diretrizes

A etapa final da metodologia consistiu na definição das diretrizes de escolha arquitetural. Essas diretrizes foram elaboradas a partir da integração entre evidências teóricas e empíricas, visando auxiliar desenvolvedores na seleção da arquitetura mais adequada a diferentes contextos de projeto no ecossistema iOS.

Este estudo não contempla medições quantitativas de desempenho, como consumo de CPU, memória ou energia, nem a implementação prática de múltiplas aplicações equivalentes utilizando cada arquitetura. Essas abordagens, embora relevantes, extrapolam o escopo e os objetivos deste trabalho, que se concentra na análise qualitativa e comparativa fundamentada na literatura e nas percepções de desenvolvedores.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os fundamentos teóricos necessários para a compreensão das arquiteturas de software analisadas neste trabalho. Inicialmente, são discutidos os conceitos centrais da arquitetura de software, seus princípios estruturais e sua relevância para a organização, manutenção e evolução de sistemas, com ênfase no contexto educacional e profissional. Em seguida, são abordadas as particularidades do desenvolvimento mobile e do ecossistema iOS, considerando suas restrições técnicas, frameworks nativos e práticas adotadas pela plataforma. A partir desse embasamento, são descritas as arquiteturas MVC, MVVM e VIP-C, destacando sua estrutura, seus fluxos de comunicação, seus benefícios e limitações. Essa fundamentação fornece a base conceitual necessária para a análise comparativa apresentada nos capítulos seguintes, permitindo avaliar de forma consistente as implicações arquiteturais de cada abordagem no desenvolvimento de aplicações iOS.

2.1 Arquitetura de Software

Segundo Bass, Clements e Kazman (2013), a arquitetura de software descreve a estrutura de um sistema, incluindo seus elementos, as propriedades relevantes desses elementos e os relacionamentos entre eles. Essa concepção destaca a arquitetura como um modelo de alto nível que orienta a organização dos componentes e suas interações, estabelecendo princípios que guiam a evolução do sistema ao longo do tempo.

Pressman e Maxim (2020) destacam que a arquitetura estabelece padrões para a organização do código e direciona o comportamento de alto nível do software. Ao segregar as

responsabilidades em diferentes agentes e promover a modularidade, a arquitetura contribui para a criação de sistemas mais compreensíveis, facilitando a manutenção. Além disso, a arquitetura atua como um mecanismo essencial de controle da complexidade em sistemas modernos, ao definir abstrações e limites estruturais que reduzem a propagação de mudanças e evitam que modificações localizadas gerem impactos indesejados em outras partes do sistema (BASS; CLEMENTS; KAZMAN, 2013; SOMMERVILLE, 2019).

No contexto da Engenharia de Software, problemas estruturais podem ser compreendidos como decisões arquiteturais recorrentes que, embora não representem defeitos funcionais imediatos, impactam negativamente atributos de qualidade ao longo do ciclo de vida do sistema. Garcia et al. (2009) definem esses problemas como *architectural bad smells*, caracterizados por combinações inadequadas de componentes, conectores e responsabilidades que reduzem propriedades como compreensibilidade, testabilidade, extensibilidade e reutilização.

Refatoração refere-se ao processo de modificar a estrutura interna do software com o objetivo de melhorar sua legibilidade, organização e manutenibilidade, sem alterar seu comportamento externo (Fowler, 1999). É um importante princípio, pois todo software que necessita de manutenção após sua criação, aplicará o conceito de refatoração para realizar as melhorias necessárias para manter o sistema em pleno funcionamento.

Entre os princípios associados a arquiteturas bem projetadas, destacam-se o baixo acoplamento, a alta coesão, a modularização e a definição precisa das responsabilidades de cada componente, princípios discutidos na literatura de Engenharia de Software (Sommerville, 2019; Fowler, 2002). Sommerville (2019) destaca que tais princípios permitem organizar sistemas complexos em estruturas compreensíveis, facilitando a reutilização de componentes e a manutenção progressiva do software. Fowler (2002) afirma que a adoção consistente desses princípios reduz a ocorrência de problemas estruturais e melhora a sustentabilidade do código ao longo do ciclo de vida do software.

A adoção de uma arquitetura adequada também desempenha papel relevante no alinhamento das equipes de desenvolvimento. Bass, Clements e Kazman (2013) destacam que a arquitetura de software atua como um importante mecanismo de comunicação e coordenação técnica entre os membros da equipe, ao definir responsabilidades, padronizar

estruturas e fornecer uma visão comum do sistema. Na ausência desse direcionamento, é possível o surgimento de retrabalho, perda de tempo em tarefas de correção estrutural e dificuldades de integração entre módulos desenvolvidos por diferentes membros do grupo. Nesse sentido, uma arquitetura bem definida contribui para mitigar esses problemas, ao promover consistência técnica, padronização do código e redução do esforço necessário para ajustes estruturais ao longo do desenvolvimento.

O impacto prático da escolha ideal de uma arquitetura torna-se ainda mais evidente quando se considera o custo de manutenção e evolução de sistemas. A maior parte do esforço total de desenvolvimento é direcionado a atividades pós-entrega, como correções, adaptações e adição de novas funcionalidades (Pressman; Maxim, 2020).

Diante desses fatores, compreender os fundamentos da arquitetura de software é essencial para analisar, comparar e selecionar padrões arquiteturais adequados às necessidades específicas de cada aplicação. Essa compreensão torna-se ainda mais relevante no desenvolvimento mobile, onde restrições de recursos, ciclos rápidos de atualização e interfaces dinâmicas exigem arquiteturas capazes de garantir organização estrutural, eficiência operacional e flexibilidade para acompanhar a evolução do produto ao longo do tempo.

2.1.1 Critérios de Comparação Arquitetural

A análise comparativa entre arquiteturas de software requer a definição prévia de critérios capazes de evidenciar diferenças estruturais, organizacionais e práticas entre os modelos estudados neste trabalho. No contexto do desenvolvimento de aplicações mobile para iOS, esses critérios permitem avaliar como decisões arquiteturais impactam diretamente a organização do código, a evolução do sistema e a sustentabilidade do produto ao longo do tempo. Neste trabalho, a comparação entre MVC, MVVM e VIP-C é conduzida com base nos critérios de modularização, fluxo de dados, acoplamento e coesão, testabilidade e manutenibilidade, muito discutidos na literatura de Engenharia de Software (Sommerville, 2019; Pressman; Maxim, 2020; Bass; Clements; Kazman, 2013)

.

2.1.2 Modularização

No contexto da arquitetura de software, a modularização refere-se ao processo de

decompor um sistema em partes menores e independentes, cada uma com responsabilidades bem definidas, de modo a facilitar a manutenção, extensibilidade e reutilização (Thaiya; Korongo; Mbugua; et al., 2022). Segundo Sommerville (2019), sistemas bem modularizados permitem que alterações sejam realizadas de forma localizada, evitando o impacto sobre outras entidades do software. Em termos arquiteturais, a modularização contribui para a clareza estrutural, incentiva a reutilização de componentes e melhora a organização do código.

Em aplicações iOS que evoluem rapidamente, a modularização é essencial para permitir a incorporação de novas funcionalidades sem comprometer a estabilidade do sistema. Arquiteturas que favorecem módulos bem definidos tendem a apresentar maior facilidade de manutenção e melhor adaptação a mudanças de requisitos, pois, por serem independentes, cada módulo pode ser reaproveitado ou expandido sem interferir nos seus pares, características desejáveis em projetos que passam por ciclos frequentes de atualização.

2.1.3 Fluxo de Dados

Bass, Clements e Kazman (2013) referem-se ao fluxo de dados como o caminho pelo qual os dados ou informações circulam entre os componentes do sistema. É uma forma de modelar como as entradas se transformam em saídas através do sistema. Portanto, o fluxo de dados bem definido torna o comportamento do sistema mais rastreável e auxilia a análise do caminho percorrido pelas informações, desde a entrada do usuário até a apresentação dos resultados.

Em aplicações iOS, cujas interfaces interativas resultam em atualizações constantes de estado, a previsibilidade do fluxo de dados contribui para a estabilidade da aplicação. Arquiteturas que organizam explicitamente esse fluxo tendem a reduzir efeitos colaterais, tais como: inconsistência na interface gráfica (um botão com título desatualizado), condições de corrida (um contador de notificações não atualizado após clique do usuário) ou dados inconsistentes (um app de mensagens que não exibe novas mensagens recebidas), tornando a rastreabilidade dos dados mais fácil e compreensível pelos desenvolvedores, especialmente em projetos colaborativos.

2.1.4 Acoplamento e Coesão

Acoplamento e coesão são conceitos complementares utilizados para avaliar a qualidade estrutural de um sistema, entendida como o conjunto de propriedades internas relacionadas à organização, à distribuição de responsabilidades e às dependências entre seus componentes, as quais influenciam diretamente a manutenibilidade e a capacidade de evolução do software (Garcia et al., 2009).

O acoplamento refere-se ao grau de dependência entre módulos que compõem a arquitetura, sendo menores graus mais vantajosos, a fim de reduzir impactos colaterais decorrentes de modificações (Pressman; Maxim, 2020). A coesão, por sua vez, indica o quão bem definidas e relacionadas são as responsabilidades internas de um componente, refletindo o grau de unidade funcional de seus elementos internos (Pressman; Maxim, 2020).

Uma vez que o grau de acoplamento refere-se a dependência entre módulos, fortes acoplamentos podem transformar simples alterações em grandes mudanças, devido à necessidade de refatorar uma quantidade excessiva de módulos. Arquiteturas que promovem separação clara de responsabilidades favorecem a coesão interna dos componentes e reduzem dependências desnecessárias, resultando em maior estabilidade estrutural e facilidade de refatoração.

2.1.5 Testabilidade

Segundo Pressman e Maxim (2020), testabilidade corresponde ao “esforço necessário para testar um programa de modo a garantir que ele desempenhe a função pretendida” (Pressman; Maxim, 2020, p. 417). No contexto arquitetural, Sommerville (2011) destaca que arquiteturas que separam claramente a lógica de negócio da interface facilitam a criação de testes unitários e reduzem a dependência de componentes gráficos durante a validação do comportamento do sistema.

No desenvolvimento de software, a testabilidade constitui um atributo de qualidade associado à facilidade com que o software pode ser analisado, modificado e validado por meio de testes. De acordo com Pressman e Maxim (2020), no modelo de qualidade da norma ISO/IEC 9126, a testabilidade é tratada como um subatributo da manutenibilidade, estando relacionada à facilidade de análise, à realização de mudanças e à verificação do comportamento do sistema. Nesse contexto, a testabilidade assume papel central na garantia de que o software apresente comportamento consistente com os requisitos definidos,

especialmente em sistemas que envolvem múltiplos estados, eventos assíncronos e interações complexas, como aplicações móveis.

2.1.6 Manutenibilidade

A manutenibilidade é um atributo de qualidade que expressa a capacidade de um software ser modificado para correção de defeitos, melhoria de desempenho ou adaptação a novos requisitos. De acordo com a norma ISO/IEC 2510, a manutenibilidade está diretamente relacionada à facilidade de análise, à realização de modificações e à verificação do software após alterações (ISO/IEC, 2011). Sommerville (2011) acrescenta que a manutenibilidade é fortemente influenciada pelas decisões arquiteturais adotadas, uma vez que a organização estrutural do sistema determina o impacto e o esforço necessário para a realização de mudanças ao longo de seu ciclo de vida.

Em aplicações mobile, onde mudanças são frequentes e exigidas por atualizações de sistema operacional, diretrizes de design ou demandas de mercado, a manutenibilidade torna-se um fator chave para a sustentabilidade do produto. Arquiteturas bem estruturadas reduzem o esforço necessário para implementar ajustes e melhorias, permitindo que o software evolua de forma consistente e controlada.

2.2 Arquiteturas Mobile e o Ecossistema iOS

O desenvolvimento de aplicações para as plataformas mobile apresentam desafios próprios que distinguem esse ambiente de outros domínios da Engenharia de Software. Dispositivos móveis possuem uma vasta variedade de oferta de hardware, mas existem restrições específicas relacionadas ao processamento, consumo de energia, conectividade, uso de memória e ciclos rápidos de atualização, fatores que definem diretamente a forma como o sistema é projetado. De acordo com Sommerville (2019), sistemas executados em plataformas com recursos limitados demandam arquiteturas capazes de organizar responsabilidades de maneira clara, de modo a minimizar o impacto de mudanças e garantir previsibilidade no comportamento da aplicação.

No ecossistema iOS, essas demandas assumem contornos particulares. A arquitetura interna das aplicações para iPhone, iPad, Apple Watch entre outros, é fortemente influenciada pelos frameworks disponibilizados pela Apple, especialmente *UIKit* e, mais recentemente,

SwiftUI. O UIKit, introduzido nas primeiras versões do iOS, baseia-se em um modelo imperativo e orientado a eventos, no qual o desenvolvedor controla explicitamente o ciclo de vida das telas, a atualização da interface e a resposta às interações do usuário. Essa abordagem influenciou diretamente a consolidação do padrão MVC como arquitetura base no desenvolvimento iOS (Apple, 2018).

Em contraste, o SwiftUI, lançado em 2019, introduziu um paradigma declarativo para a construção de interfaces, no qual a interface passa a ser descrita como uma função do estado da aplicação. Nesse modelo, mudanças nos dados provocam automaticamente a atualização da interface, exigindo arquiteturas que favoreçam separação clara de responsabilidades, fluxo de dados previsível e gerenciamento explícito de estado. Essa mudança ampliou a necessidade de arquiteturas mais bem definidas, capazes de sustentar aplicações reativas e evolutivas no ecossistema iOS (Apple, 2025)

Esses frameworks definem não apenas o ciclo de vida das interfaces, mas também o modo como camadas de apresentação, lógica e navegação interagem e são expostas aos desenvolvedores. Desde o lançamento do *Swift* em 2014, a linguagem consolidou práticas modernas de programação, como tipagem segura, orientação a protocolos e modularização, o que ampliou a necessidade de arquiteturas adequadas para organizar fluxos de dados e estruturas de responsabilidade (Apple, 2023).

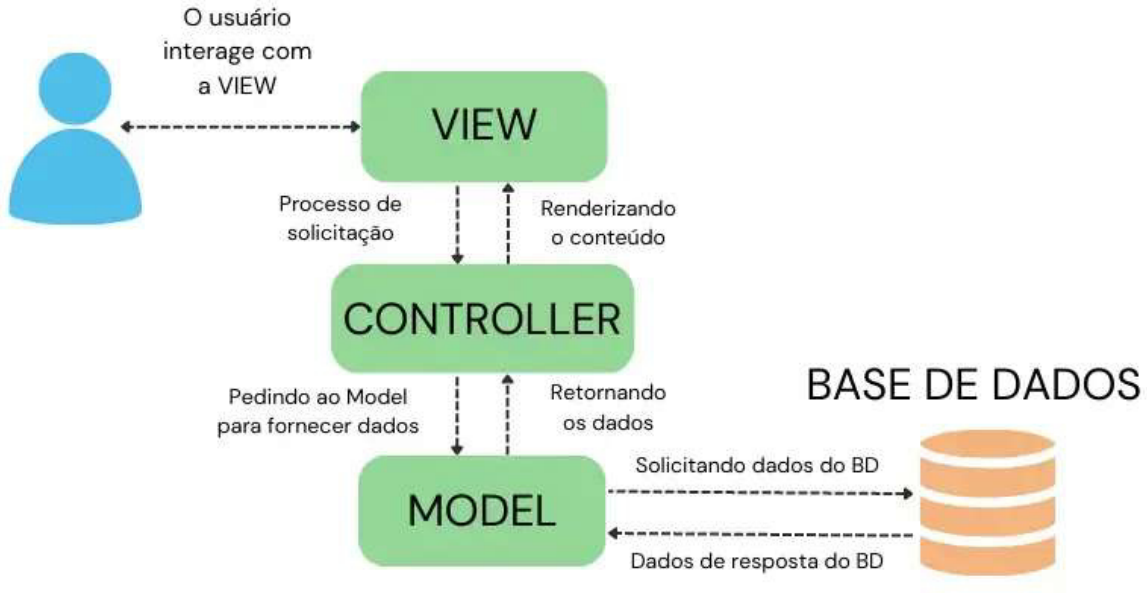
O desenvolvimento para o ecossistema iOS exige que as aplicações lidem com eventos assíncronos, atualizações constantes de estado, múltiplas telas e grande volume de interação entre componentes da interface. Assim, é necessário delimitar a separação entre a lógica de apresentação e a lógica de domínio, para melhorar o entendimento do código e facilitar testes automatizados. Além disso, a divisão de agentes com responsabilidades distintas permite que novas funcionalidades sejam incorporadas sem comprometer a estrutura já existente.

Assim, a definição de uma arquitetura adequada para aplicações iOS não se limita a uma escolha estilística, mas constitui uma decisão técnica essencial para garantir organização, estabilidade e escalabilidade. A compreensão das particularidades do ambiente mobile e dos frameworks oficiais da plataforma fornece a base conceitual necessária para o estudo das arquiteturas MVC, MVVM e VIP-C, que são apresentadas nas seções seguintes.

2.3 Arquitetura MVC (Model–View–Controller)

A arquitetura Model–View–Controller (MVC) é um padrão arquitetural consolidado e tradicional no desenvolvimento de software para sistemas interativos (Sommerville, 2019). Proposta originalmente por Trygve Reenskaug no final da década de 1970, durante seu trabalho no Xerox Palo Alto Research Center (Xerox PARC) (Reenskaug, 1979), essa arquitetura tem como princípio a separação de responsabilidades entre modelo de dados, apresentação (View) e controle da interação, promovendo uma organização estrutural que facilita a compreensão, a manutenção e a evolução do sistema ao longo do tempo (Sommerville, 2019).

Figura 2 – Arquitetura Model–View–Controller (MVC)



Fonte: Normando (2024). (adaptado pelo autor).

No modelo ilustrado na Figura 2, o Model é responsável por representar os dados e as regras de negócio da aplicação, incluindo validações e estados internos. A View corresponde à camada de apresentação, encarregada exclusivamente de exibir informações e capturar interações do usuário, como cliques, gestos e mudanças de orientação da interface. O Controller atua como intermediário entre Model e View, recebendo eventos da interface, coordenando o fluxo de dados e acionando atualizações necessárias. Essa organização permite que mudanças em uma camada tenham impacto reduzido sobre as demais, princípio fundamental da Engenharia de Software orientada à manutenção (Sommerville, 2019).

No contexto do desenvolvimento iOS, a MVC consolidou-se como arquitetura base devido à própria estrutura do framework UIKit. A Apple organiza o ciclo de vida das telas a partir de controladores de interface (*UIViewController*), que assumem naturalmente o papel de Controller no padrão MVC. De acordo com a documentação oficial da Apple (Apple, 2023), o controlador gerencia eventos de interação, coordena atualizações da interface e estabelece a comunicação entre os dados do modelo e a visualização apresentada ao usuário. Essa aderência direta ao framework torna o MVC uma escolha intuitiva para o início de projetos iOS.

Entre as principais vantagens da arquitetura MVC, destaca-se a melhoria na organização do código proporcionada pela separação entre Model, View e Controller, o que contribui para uma compreensão mais clara da estrutura geral da aplicação e facilita a identificação de erros e a implementação de modificações em sistemas existentes (GeeksforGeeks, 2024). Além disso, a arquitetura MVC favorece o desenvolvimento paralelo, permitindo que diferentes desenvolvedores atuem simultaneamente sobre o modelo, a visão e o controlador sem interferência direta entre si, bem como a reutilização de componentes, especialmente do Model, que pode ser empregado em diferentes partes da aplicação, reforçando a eficiência do processo de desenvolvimento (GeeksforGeeks, 2024).

Apesar de suas vantagens, a aplicação prática do MVC no desenvolvimento iOS apresenta limitações discutidas na literatura técnica e na comunidade profissional. Devido ao papel centralizador do Controller, é comum que essa camada acumule responsabilidades excessivas, concentrando lógica de apresentação, regras de interface e até parte da lógica de negócio (Clean Swift, 2019). Esse fenômeno é conhecido como *Massive View Controller*, um antipadrão recorrente em projetos iOS.

Rochabrun (2020) demonstra que a ausência de uma separação mais rigorosa de responsabilidades leva à criação de controladores extensos, difíceis de testar, manter e evoluir. Esse acúmulo compromete a testabilidade, aumenta o acoplamento entre componentes e torna alterações simples propensas a efeitos colaterais.

2.4 Arquitetura Model–View–ViewModel (MVVM)

A arquitetura Model–View–ViewModel (MVVM) surgiu como um padrão arquitetural voltado à separação clara entre interface gráfica e lógica de apresentação, com o objetivo de

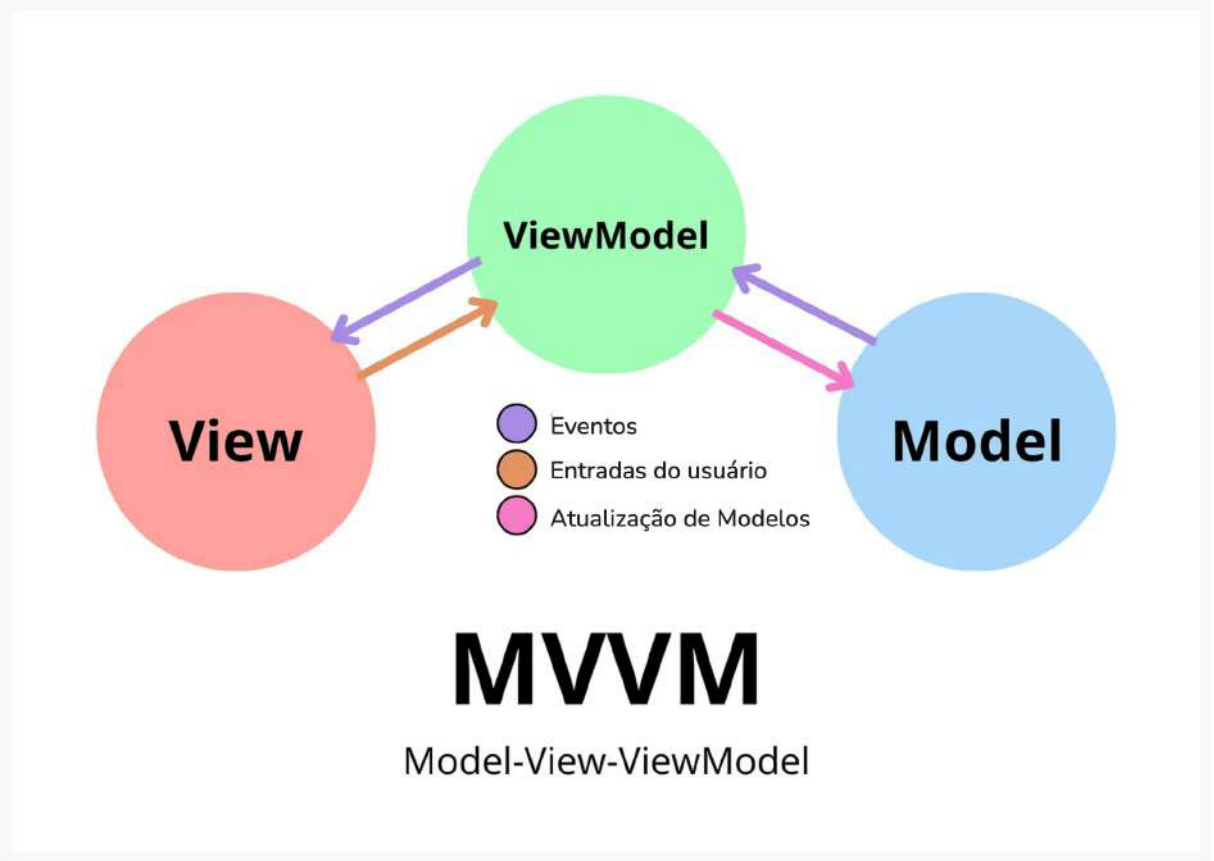
reduzir o acoplamento entre camadas e favorecer a testabilidade (Gossman, 2005). O padrão foi formalizado por Gossman (2005) no contexto do desenvolvimento de interfaces declarativas na plataforma Windows Presentation Foundation (WPF), propondo uma estrutura em que a View apenas reflete o estado exposto pelo ViewModel, enquanto este concentra a lógica de apresentação e a comunicação com o modelo. Segundo o autor, essa organização busca facilitar a manutenção do código e permitir a validação da lógica da aplicação independentemente da interface gráfica (Gossman, 2005).

A literatura técnica aponta que a MVVM tornou-se popular especialmente em cenários nos quais o uso da MVC tradicional levou ao acúmulo excessivo de lógica na camada de controle, dificultando a manutenção e a evolução do software (Sommerville, 2019; Microsoft, 2023). Assim, a adoção do MVVM ocorre como uma alternativa arquitetural que busca mitigar problemas recorrentes observados em arquiteturas mais centralizadas, sem abandonar os princípios clássicos de organização estrutural.

A estrutura do MVVM, assim como no MVC, é composta por três elementos centrais: Model, View e ViewModel (Microsoft, 2023). O Model representa os dados e as regras de negócio da aplicação, sendo responsável por manter o estado e executar operações essenciais do domínio. A View é encarregada exclusivamente da interface gráfica e da interação com o usuário, limitando-se à exibição das informações e à captura de eventos. O ViewModel atua como camada intermediária, concentrando a lógica de apresentação e transformando os dados do modelo em informações prontas para exibição. Essa organização promove maior independência entre componentes, evitando que a View acesse diretamente o Model e contribuindo para uma estrutura mais modular e organizada (Gossman, 2005).

A relação entre os componentes e o fluxo conceitual do padrão MVVM pode ser observada na Figura 3, que ilustra a separação clara de responsabilidades e a comunicação indireta entre a interface e a lógica de domínio.

Figura 3 – Arquitetura Model–View–ViewModel (MVVM)



Fonte: CARDOSO, Gustavo. Medium (2021) (adaptado pelo autor).

O fluxo de comunicação entre View e ViewModel ocorre por meio de mecanismos de observação ou atualização de estado, nos quais a interface é notificada quando propriedades expostas pelo ViewModel sofrem alterações. Segundo Gossman (2005), o ViewModel atua como uma abstração da View, expondo dados e comportamentos de forma que a interface possa reagir automaticamente às mudanças de estado, sem dependência direta da lógica de domínio. Conceitualmente, a interface deve refletir o estado fornecido pelo ViewModel, mantendo a lógica de apresentação desacoplada da camada visual, o que resulta em um fluxo de dados mais previsível e em melhor organização estrutural (Microsoft, 2023).

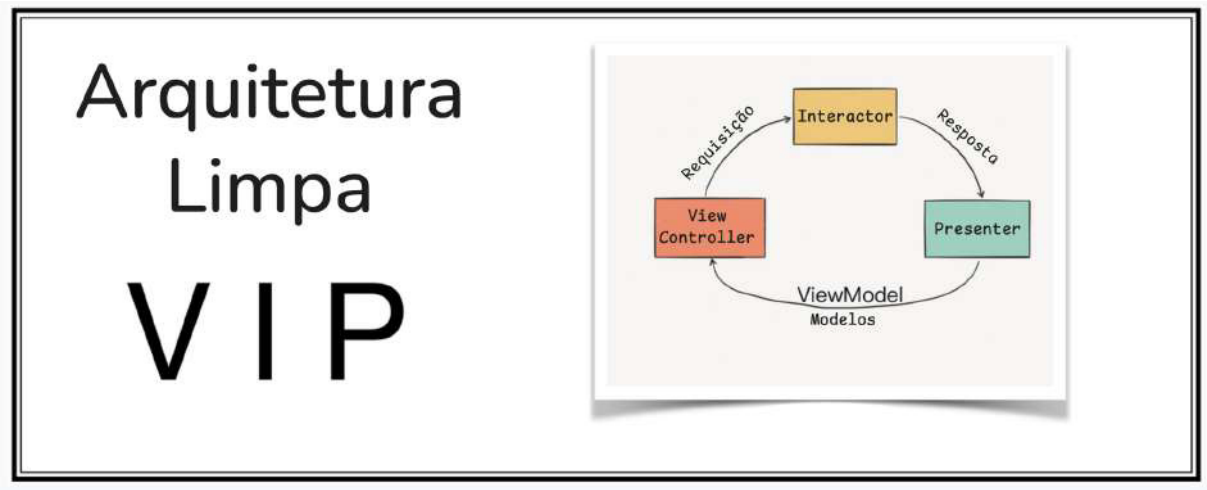
Entre as principais vantagens do MVVM, destaca-se a melhoria da testabilidade, uma vez que o ViewModel pode ser testado de forma independente da interface gráfica (Microsoft, 2024). A separação entre lógica de apresentação e camada visual reduz o acoplamento e minimiza o risco de que alterações na interface impactem diretamente o comportamento interno da aplicação.

2.5 Arquitetura VIP-C (View, Interactor, Presenter e Coordinator)

A arquitetura VIP-C (*View–Interactor–Presenter–Coordinator*) é uma variação arquitetural adotada no ecossistema iOS que combina a estrutura básica do padrão VIP (*View–Interactor–Presenter*) com o uso do *Coordinator* como mecanismo de organização dos fluxos de navegação. Essa abordagem não constitui um padrão formal definido por um autor específico, mas emerge da prática da comunidade de desenvolvimento iOS como uma adaptação arquitetural voltada à separação clara de responsabilidades e à organização do fluxo entre camadas. Materiais técnicos utilizados na comunidade, como a documentação do Clean Swift, descrevem o uso do VIP e suas variações como estratégias para reduzir acoplamento, melhorar testabilidade e evitar o acúmulo excessivo de responsabilidades em controladores de interface (Clean Swift, 2019). Além disso, embora não tenha sido formalmente proposta por Robert C. Martin, a organização estrutural do VIP-C dialoga com princípios da Arquitetura Limpa, especialmente no que se refere à independência entre camadas e à centralização das regras de negócio em componentes bem definidos (Martin, 2018).

Diferentemente de arquiteturas mais simples, o VIP-C organiza o software em camadas bem definidas, com responsabilidades explicitamente separadas, o que favorece modularização, testabilidade e clareza estrutural (Miciano, 2021). Segundo Miciano (2021), essa abordagem pode ser adotada por equipes que buscam maior controle sobre a organização do código e sobre o fluxo entre interface, lógica de negócio e navegação.

Figura 4 – Diagrama da arquitetura iOS VIP conforme Miciano (2021)



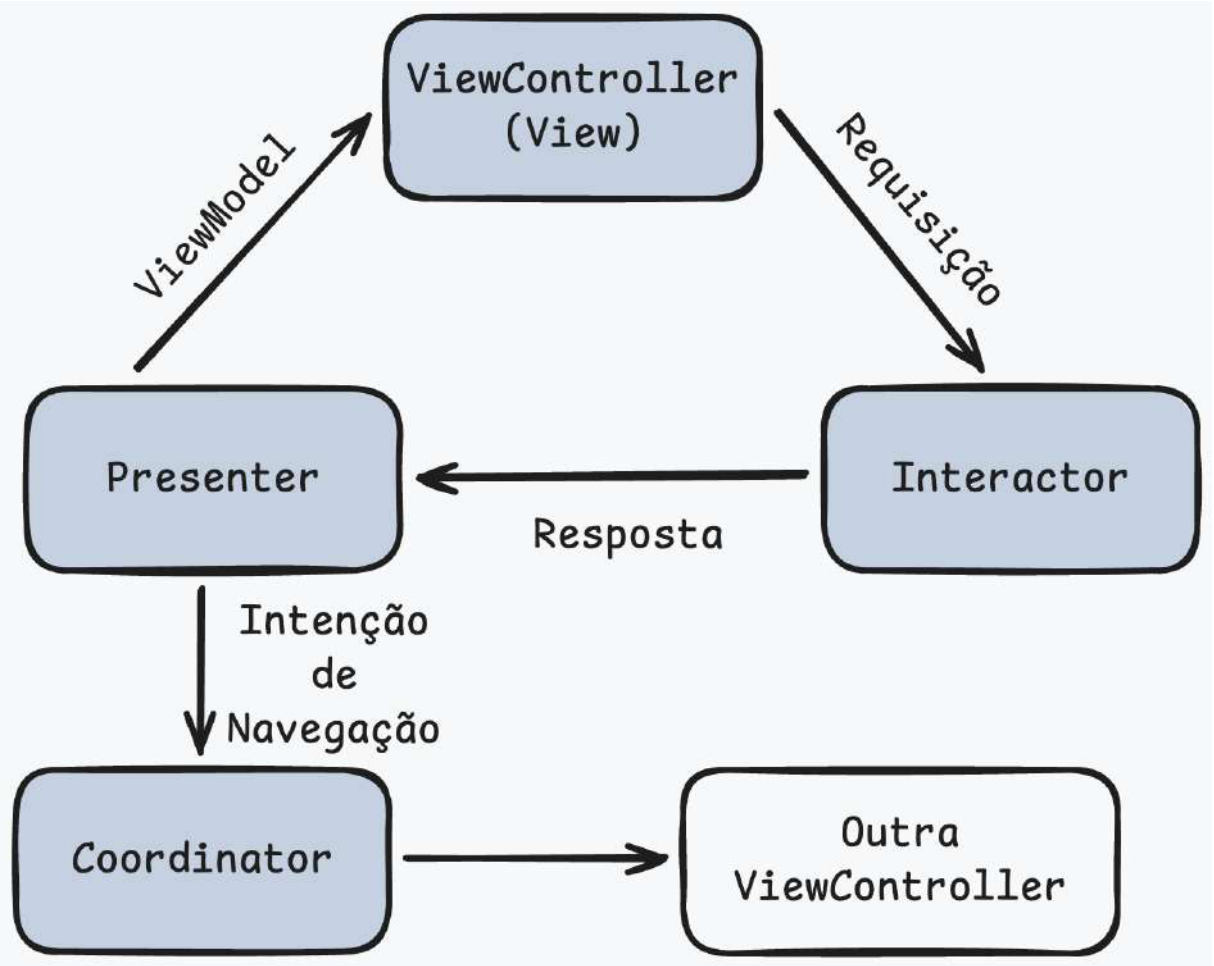
Fonte: MICIANO, Fábio. Medium (2021) (adaptado pelo autor).

A estrutura base pode ser definida como a representada na Figura 4. A View, geralmente representada por um UIViewController, tem como responsabilidade exclusiva a exibição da interface gráfica e a captura das interações do usuário (Miciano, 2021). Ela não implementa regras de negócio nem toma decisões complexas sobre o comportamento da aplicação, mantendo-se restrita à camada de apresentação. Em arquiteturas baseadas em VIP, a View atua como um ponto de entrada de eventos, encaminhando as ações do usuário para a camada responsável pelo processamento do caso de uso (Clean Swift, 2023; Miciano, 2021).

O Interactor é responsável pelo processamento da lógica associada ao caso de uso. Essa camada manipula dados, aciona serviços, aplica regras de negócio e executa operações necessárias para atender às solicitações provenientes da interface, mantendo-se isolada de preocupações relacionadas à apresentação visual. Essa separação favorece a testabilidade e a independência das regras de negócio em relação à interface gráfica (Clean Swift, 2023; Law, 2019).

O Presenter atua como intermediário entre a lógica de negócio e a interface. Sua função é transformar os dados fornecidos pelo Interactor em estruturas adequadas para exibição, comumente denominadas *ViewModels*. O Presenter organiza informações, formata dados, valida estados e prepara o conteúdo a ser apresentado ao usuário, sem depender diretamente de componentes visuais. A View consome esses dados e atualiza seus elementos gráficos sem conhecer a lógica interna que os originou. O fluxo principal da arquitetura segue o padrão View → Interactor → Presenter → View, promovendo previsibilidade, rastreabilidade e clareza no comportamento do sistema (Clean Swift, 2023; Miciano, 2021).

Figura 5 – Diagrama conceitual da arquitetura VIP-C com Coordinator



Fonte: Elaborado pelo autor (2026).

O fluxo de navegação proposto pela arquitetura VIP-C é componente adicional à parte e pode ser observado na Figura 5. O *Coordinator* ocupa um papel complementar e estratégico na arquitetura VIP-C, assumindo a responsabilidade pela navegação da aplicação e pela composição dos módulos arquiteturais. Em vez de acoplar transições de tela diretamente aos controladores de interface ou ao *Presenter*, o *Coordinator* centraliza a definição dos fluxos de navegação, permitindo organizar caminhos e transições de forma independente das camadas de apresentação e lógica de negócio. Essa abordagem é discutida na literatura técnica voltada ao desenvolvimento iOS, na qual o *Coordinator Pattern* é apresentado como uma solução para reduzir o acoplamento entre telas, melhorar a legibilidade do código e facilitar a manutenção de aplicações com fluxos complexos de navegação (Khanlou, 2015; Law, 2019; Clean Swift, 2023).

A separação rigorosa entre interface, regras de negócio, apresentação e navegação favorece a escrita de testes unitários, principalmente nas camadas de Interactor e Presenter, que podem ser testadas independentemente da interface gráfica. Além disso, a arquitetura facilita a paralelização do trabalho em equipes, permitindo que desenvolvedores atuem em camadas distintas sem interferência direta entre si.

Ao centralizar a responsabilidade pela navegação, o uso do *Coordinator* permite separar a definição dos fluxos de tela das camadas de apresentação e lógica de negócio, reduzindo a dependência direta entre controladores e caminhos de navegação. Essa separação estrutural torna a arquitetura mais flexível do ponto de vista organizacional, facilitando ajustes e reorganizações nos fluxos da aplicação sem a necessidade de alterações extensas nas demais camadas.

Apesar de seus benefícios estruturais, a adoção do VIP-C pode demandar um nível maior de organização e complexidade inicial, especialmente quando comparada a arquiteturas mais simples como MVC ou MVVM, devido à maior quantidade de camadas, ao *boilerplate* e à curva de aprendizado associada ao padrão (Miciano, 2021). A composição dos módulos demanda a definição de múltiplos componentes, como *View*, *Interactor*, *Presenter* e *Coordinator*, além do uso de protocolos e abstrações para comunicação entre camadas. Esse nível de segmentação pode representar um esforço adicional nas fases iniciais do desenvolvimento.

Em síntese, a arquitetura VIP-C destaca-se pela clareza na definição dos papéis de cada componente e pela previsibilidade do fluxo de dados e de navegação. A compreensão de sua estrutura e de seus elementos constitutivos é fundamental para que, posteriormente, seja possível analisá-la de forma comparativa em relação a outras arquiteturas adotadas no ecossistema iOS, considerando diferentes critérios e contextos de aplicação.

3 TRABALHOS RELACIONADOS

Os trabalhos apresentados analisaram padrões arquiteturais aplicados ao desenvolvimento de aplicações móveis, oferecendo base conceitual e prática para estudos comparativos. Essas pesquisas abordam, principalmente, questões de modularidade, organização do código, desempenho e facilidade de manutenção, aspectos que também orientam a análise proposta neste trabalho.

Correia (2025) realizou um estudo comparativo entre as arquiteturas MVC e MVVM utilizando o framework Flutter, avaliando métricas como consumo de CPU, memória e tempo de renderização, cuja metodologia adotada consistiu no desenvolvimento de uma mesma aplicação funcional em duas versões arquiteturais distintas, submetidas às mesmas condições de teste em ambiente controlado. Seus resultados demonstram que o MVVM apresenta melhor eficiência e estruturação interna, reforçando a relevância da separação clara de responsabilidades para a evolução de sistemas móveis de plataforma híbrida utilizando Flutter.

Pereira e Souza (2023) desenvolveram um estudo orientado ao uso de arquiteturas de software no contexto mobile, discutindo padrões como MVC, Portas e Adaptadores e Clean Architecture. Os autores destacam que padrões com camadas bem definidas promovem maior previsibilidade, facilidade de manutenção e capacidade de evolução contínua, embora demandem maior planejamento inicial.

Barbosa (2022) analisou comparativamente os padrões MVC, MVP, MVVM e MVI na plataforma Android, utilizando técnicas como SAMM (*Software Architecture Analysis Method*) para avaliar atributos de qualidade como escalabilidade e clareza estrutural. Os resultados apontam que MVVM oferece melhor equilíbrio entre modularidade e esforço de implementação, tornando-se um padrão adequado para equipes que precisam de organização interna consistente. Embora voltado ao Android, o estudo aborda desafios que também ocorrem no ecossistema iOS, especialmente no tocante à separação entre interface, lógica de apresentação e regras de negócio.

Souza(2020) apresenta ainda um trabalho relacionado ao desenvolvimento de um aplicativo iOS para gestão de eventos, destacando como a escolha arquitetural influencia diretamente práticas de qualidade, fluxo de navegação e suporte a testes automatizados. O estudo reforça que, no desenvolvimento nativo para iOS, arquiteturas mais bem definidas contribuem para reduzir o acoplamento e facilitar tanto o desenvolvimento quanto a manutenção do software.

Além desses trabalhos acadêmicos, há também contribuições relevantes da comunidade técnica. Clean Swift (2019) discute o problema conhecido como *Massive View Controller* e propõe a arquitetura VIP/VIP-C como alternativa para organizar de forma mais

clara o fluxo entre visão, lógica de negócio e apresentação. Esse material é referenciado por desenvolvedores iOS e apresenta uma abordagem aplicada para modularização e redução de acoplamento, mantendo organização previsível entre os componentes.

A literatura analisada indica convergência quanto aos benefícios da modularização para a manutenção, a evolução e a clareza estrutural dos sistemas, apesar do aumento da complexidade inicial, conforme discutido por Souza (2020), Barbosa (2022) e Pereira e Souza (2023). Entretanto, observa-se que há poucos trabalhos focados exclusivamente no ecossistema iOS com comparação direta entre arquiteturas amplamente utilizadas na prática como MVC, MVVM e VIP-C especialmente sob a ótica da curva de aprendizado, organização de camadas e adequação ao desenvolvimento inicial de produtos. Assim, este trabalho contribui para preencher essa lacuna ao realizar uma análise comparativa estruturada especificamente dentro do contexto iOS, considerando as implicações práticas dessas arquiteturas nos estágios iniciais do desenvolvimento mobile.

A Quadro 1 apresenta uma síntese comparativa dos principais trabalhos relacionados discutidos nesta seção, destacando o contexto de aplicação, o sistema operacional, os critérios investigados e as arquiteturas analisadas em cada estudo.

Este trabalho se destaca em relação aos trabalhos analisados, pois este estudo não se concentra na avaliação de desempenho computacional nem na implementação de aplicações equivalentes em múltiplas arquiteturas, mas propõe uma análise comparativa integrada, fundamentada simultaneamente em revisão sistemática da literatura e em evidências empíricas descritivas coletadas junto a desenvolvedores iOS.

Diferentemente das abordagens que privilegiam métricas específicas ou plataformas distintas, este trabalho foca exclusivamente no ecossistema iOS e na comparação entre arquiteturas MVC, MVVM e VIP-C sob a perspectiva de atributos de qualidade estrutural, organização arquitetural e definição de qual mais adequada a depender do contexto de cada tipo de projeto. Como resultado, o estudo avança oferecendo suporte conceitual e prático para decisões arquiteturais em diferentes estágios de desenvolvimento de aplicações iOS.

Quadro 1 – Comparação entre trabalhos relacionados sobre arquiteturas de software.

Trabalho	Contexto de aplicação	Sistema Operacional	Critérios investigados	Arquiteturas investigadas	Metodologia
Correia (2025)	Desenvolvimento de aplicações móveis utilizando framework multiplataforma	Android e iOS (Flutter)	Desempenho (CPU, memória), tempo de renderização, organização estrutural	MVC, MVVM	Estudo experimental comparativo
Pereira e Souza (2023)	Aplicações mobile orientadas a boas práticas de Engenharia de Software	Mobile (plataforma genérica)	Modularidade, previsibilidade, manutenção, separação de camadas	MVC, Portas e Adaptadores, Clean Architecture	Análise qualitativa da estrutura do código
Barbosa (2022)	Desenvolvimento nativo de aplicações móveis	Android	Escalabilidade, clareza estrutural, esforço de implementação (SAAM)	MVC, MVP, MVVM, MVI	Estudo de caso
Souza(2020)	Aplicativo iOS para gestão de eventos	iOS	Qualidade do código, fluxo de navegação, suporte a testes automatizados	Arquitetura em camadas (não especificada formalmente)	Estudo de caso
Este trabalho	Comparação entre arquiteturas de software	iOS	Modularização, Fluxo de dados, testabilidade, manutenibili	MVC, MVVM, VIP-C	Revisão sistemática da literatura e pesquisa com o

			dade		público alvo

Fonte: Elaborado pelo autor (2026).

4 REVISÃO SISTEMÁTICA DE LITERATURA

Este capítulo apresenta uma revisão sistemática da literatura voltada à análise de arquiteturas de software aplicadas ao desenvolvimento de aplicações mobile, com ênfase no ecossistema iOS. A revisão abrange trabalhos acadêmicos e técnicos publicados em periódicos científicos, repositórios institucionais e fontes especializadas da área de Engenharia de Software, considerando arquiteturas abundantemente utilizadas na prática profissional, como MVC, MVVM e VIP-C.

O objetivo desta revisão sistemática é identificar, organizar e sintetizar evidências existentes na literatura acerca de como diferentes abordagens arquiteturais se relacionam com atributos estruturais relevantes ao desenvolvimento de software, tais como modularização, fluxo de dados, acoplamento e coesão, testabilidade e manutenibilidade. A partir desse levantamento, busca-se fundamentar e contextualizar, a partir da produção científica existente, os critérios adotados neste trabalho para a análise comparativa das arquiteturas de software no contexto mobile, evidenciando de que forma esses atributos são conceituados, discutidos e aplicados em estudos anteriores.

Adicionalmente, a revisão sistemática permite mapear os métodos de investigação empregados na literatura, identificar lacunas de pesquisa e compreender como a escolha arquitetural tem sido abordada sob diferentes perspectivas acadêmicas e profissionais. O relato detalhado do processo metodológico e dos dados obtidos nesta revisão segue as diretrizes do modelo PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) (Moher et al., 2009), adotado com o objetivo de garantir transparência, rigor metodológico e reprodutibilidade dos resultados.

4.1 Objetivos da revisão sistemática

4.1.1 Objetivo Principal

Investigar, por meio de uma revisão sistemática da literatura científica e técnica, como Arquiteturas de Software aplicadas ao desenvolvimento de aplicações mobile no ecossistema

iOS são descritas, discutidas e analisadas, com foco nas arquiteturas MVC, MVVM, VIP-C/VIPER, considerando atributos estruturais relevantes à Engenharia de Software, tais como modularização, fluxo de dados, acoplamento e coesão, testabilidade e manutenibilidade.

Essa revisão tem como objetivo responder a questão da pesquisa levantada neste trabalho.

4.1.2 Objetivos Secundários

1. Identificar as arquiteturas de software mais investigadas na literatura relacionada ao desenvolvimento de aplicações mobile no ecossistema iOS, com ênfase em MVC, MVVM e VIPER/VIP-C;
2. Levantar os principais atributos de qualidade arquitetural analisados nos estudos;
3. Analisar como a literatura define e discute atributos arquiteturais relevantes, tais como modularização, fluxo de dados, acoplamento e coesão, testabilidade e manutenibilidade;
4. Identificar benefícios e desafios reportados na adoção dessas arquiteturas em aplicações iOS.
5. Investigar em quais contextos de projeto (escopo inicial, evolução do sistema, complexidade crescente) cada arquitetura é apontada como mais adequada.

4.2 Formulação das Questões de Pesquisa

A formulação das questões de pesquisa é uma etapa central na Revisão Sistemática da Literatura, pois orienta a definição do escopo da investigação, a estratégia de busca e o processo de seleção e análise dos estudos relevantes. De acordo com Kitchenham et al. (2009), questões de pesquisa bem definidas são fundamentais para garantir foco, reprodutibilidade e consistência metodológica em revisões sistemáticas no campo da Engenharia de Software.

Neste trabalho, as questões de pesquisa foram elaboradas a partir dos objetivos da revisão sistemática e dos critérios arquiteturais adotados para a análise comparativa das arquiteturas MVC, MVVM e VIP-C.

O processo de formulação das questões seguiu diretrizes propostas por Kitchenham et al. (2009) e Brereton et al. (2007), sendo adaptado ao escopo e à natureza deste estudo, que

possui caráter qualitativo e comparativo. As questões de pesquisa foram estruturadas de modo a permitir a identificação de evidências conceituais, empíricas e técnicas na literatura, servindo como base para a síntese dos resultados da revisão e para a análise comparativa desenvolvida nos capítulos subsequentes.

As Questões de Pesquisa (QP) que nortearam esta revisão sistemática estão apresentadas no Quadro 2, juntamente com a justificativa de cada uma no contexto do presente trabalho.

Quadro 2 – Questões de pesquisa e suas justificativas.

ID	Questão de Pesquisa	Justificativa
QP1	Quais arquiteturas de software são mais recorrentes na literatura científica e técnica sobre o desenvolvimento de aplicações mobile no ecossistema iOS?	Identificar os padrões arquiteturais mais discutidos e utilizados no contexto iOS, contextualizando a seleção das arquiteturas analisadas neste trabalho.
QP2	Como a literatura descreve e caracteriza arquiteturas MVC, MVVM e VIP-C aplicadas ao desenvolvimento mobile iOS?	Compreender as principais características estruturais atribuídas a cada arquitetura, observando como se deu sua construção.
QP3	De que forma as arquiteturas MVC, MVVM e VIP-C impactam atributos de qualidade de software, como modularização, fluxo de dados, acoplamento e coesão, segundo a literatura?	Investigar como diferentes abordagens arquiteturais organizam responsabilidades e fluxos internos, bem como suas implicações estruturais.
QP4	Quais evidências a literatura apresenta sobre a relação entre arquiteturas de software e atributos como testabilidade e manutenibilidade em	Analisar como decisões arquiteturais influenciam a criação, o isolamento e a

	aplicações mobile iOS?	manutenção de testes, bem como a evolução do sistema ao longo do tempo.
QP5	Quais benefícios e desafios são relatados na literatura quanto à adoção das arquiteturas MVC, MVVM e VIP-C em projetos iOS?	Identificar vantagens, limitações e dificuldades práticas associadas à adoção dessas arquiteturas. .
QP6	Em quais contextos de projeto (aplicações iniciais, sistemas em evolução ou projetos de maior porte) a literatura indica maior adequação das arquiteturas MVC, MVVM e VIP-C?	Compreender como a escolha arquitetural é relacionada ao contexto de uso, escopo do projeto e necessidades de evolução do software.

Fonte**:** Elaborado pelo autor (2026).

4.3 Planejamento da Revisão Sistemática

O planejamento da Revisão Sistemática da Literatura (RSL) corresponde à construção do protocolo metodológico que orienta todas as etapas da investigação, desde a definição da estratégia de busca até os procedimentos de seleção, análise e síntese dos estudos. Essa etapa é fundamental para assegurar a validade dos resultados, promover transparência metodológica e reduzir vieses durante o processo de revisão.

A revisão sistemática foi planejada considerando atributos arquiteturais previamente definidos neste trabalho, os quais orientaram tanto a formulação das questões de pesquisa quanto a extração e a síntese dos dados.

De acordo com Kitchenham et al. (2009), o protocolo de uma RSL deve explicitar claramente as decisões metodológicas adotadas, permitindo que o estudo seja compreendido, avaliado e eventualmente reproduzido por outros pesquisadores.

4.3.1 Estratégia de busca

As estratégias de busca e seleção dos estudos foram definidas com base em quatro elementos fundamentais: fontes de busca, idioma, período de publicação e palavras-chave.

- Fontes de busca: Google Scholar, SciELO, Repositórios institucionais de universidades brasileiras, IEEE Xplore, ACM Digital Library, Scopus. Essas fontes foram escolhidas por sua relevância, abrangência temática e facilidade de acesso a estudos científicos e técnicos relacionados ao desenvolvimento de software e arquiteturas mobile.
- Idioma: Português e Inglês.
- Período de publicação: Não foi estabelecido um limite rígido de data.
- Palavras chave: arquitetura de software, software architecture, aplicações mobile, mobile applications, iOS, MVC, MVVM, VIPER, VIP-C, testabilidade, manutenibilidade, modularização. Esses termos foram combinados por meio de operadores booleanos para compor a expressão geral de busca utilizada nas bases de dados selecionadas.

4.3.2 Critérios e procedimentos para seleção dos estudos

Conforme destacado por Dresch et al**.** (2015), uma revisão sistemática pode estar sujeita a vieses se os critérios de seleção não forem claramente definidos e rigorosamente aplicados. Dessa forma, foram estabelecidos critérios objetivos de inclusão e exclusão, aplicados de maneira sequencial durante o processo de triagem.

Critérios de exclusão:

1. Estudos que não abordem arquiteturas de software;
2. Trabalhos fora do contexto de aplicações mobile ou sem relação com o ecossistema iOS;
3. Trabalhos focados exclusivamente em desempenho de hardware ou redes;
4. Artigos sem acesso ao texto completo;
5. Estudos duplicados entre as bases de dados;
6. Trabalhos que tratam apenas de frameworks sem discussão arquitetural.

Critérios de inclusão

1. Estudos que abordem arquiteturas de software aplicadas ao desenvolvimento mobile, incluindo iOS;
2. Trabalhos que discutam MVC, MVVM, VIPER ou VIP-C;
3. Publicações que analisem atributos arquiteturais como modularização, fluxo de dados, acoplamento, coesão, testabilidade ou manutenibilidade.

4.3.3 Expressão Geral de Busca

A expressão geral de busca foi construída a partir da combinação das palavras-chave definidas na etapa de planejamento, utilizando operadores booleanos para ampliar a cobertura da busca e, ao mesmo tempo, manter a conformidade ao escopo da pesquisa.

A expressão geral de busca foi construída de modo a combinar quatro elementos centrais:

- o domínio da pesquisa (desenvolvimento mobile e ecossistema iOS);
- o objeto de estudo (arquitetura de software);
- arquiteturas específicas relevantes (MVC, MVVM, VIPER/VIP-C);
- atributos de qualidade arquitetural relacionados à análise proposta, como modularização, acoplamento, coesão, testabilidade e manutenibilidade.

A string foi formulada em língua inglesa, por ser o idioma predominante nas palavras-chaves de publicações científicas da área, e aplicada de forma equivalente nas bases de dados selecionadas, com ajustes pontuais quando necessário para atender às particularidades de cada mecanismo de busca. A versão final da string de busca adotada nesta revisão está apresentada a seguir:

(mobile application" OR "mobile app" OR "iS application" OR iOS)

AND ("software architecture" OR "application architecture")

AND (MVC OR MVVM OR VIPER OR VIP OR "VIP-C" OR "View Interactor Presenter")

AND (modularity OR maintainability OR testability OR coupling OR cohesion OR "data flow")

4.3.4 Extração de Dados

Após a aplicação dos critérios de seleção, os estudos incluídos foram analisados por meio de leitura completa. Para cada estudo, foram extraídas informações relevantes para os objetivos da revisão, incluindo:

- arquiteturas de software abordadas;
- contexto de aplicação (mobile e/ou iOS);
- atributos arquiteturais discutidos;
- tipo de estudo (conceitual, empírico ou relato de experiência);
- principais conclusões relacionadas à arquitetura.

Para garantir a padronização e a consistência na coleta das informações relevantes dos estudos selecionados, foi elaborado um formulário de extração de dados (Quadro 3) durante a fase de planejamento da revisão sistemática. Esse formulário teve como objetivo orientar a identificação, organização e síntese das informações necessárias para responder às questões de pesquisa e subsidiar a análise comparativa das arquiteturas de software investigadas.

O formulário de extração foi estruturado de modo a contemplar dados bibliográficos, contextuais e analíticos, além de informações diretamente relacionadas aos atributos arquiteturais definidos como critérios de comparação neste trabalho. O Quadro 3 apresenta os campos considerados no processo de extração de dados.

Quadro 3 – Formulário de Extração de Dados da RSL

ID	**Descrição dos dados extraídos**
1	Identificação do estudo (autores, ano de publicação, veículo de publicação)
2	Tipo de estudo (artigo científico, estudo de caso, revisão, relato de experiência, etc.)
3	Arquiteturas de software abordadas (MVC, MVVM, VIPER/VIP-C ou outras)
4	Contexto de aplicação (iOS, mobile genérico, multiplataforma)
5	Objetivo principal do estudo

6	Descrição da organização arquitetural proposta ou analisada
7	Como o fluxo de dados é tratado ou descrito na arquitetura
8	Evidências ou discussões relacionadas à modularização
9	Evidências ou discussões relacionadas à acoplamento e coesão
10	Evidências ou discussões relacionadas à testabilidade
11	Evidências ou discussões relacionadas à manutenibilidade
12	Benefícios relatados na adoção da arquitetura
12	Desafios, limitações ou problemas reportados
14	Contextos de projeto nos quais a arquitetura é considerada mais adequada

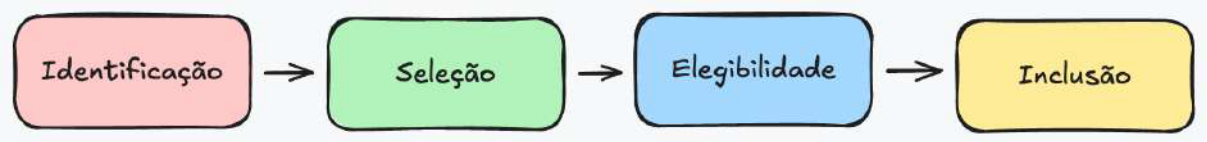
Fonte: Elaborado pelo autor (2026).

A etapa de extração de dados foi realizada no período de dezembro de 2025 a janeiro de 2026. A definição desse intervalo temporal delimita o conjunto de evidências considerado na revisão e garante a rastreabilidade do processo de extração.

4.4 Execução da Revisão Sistemática

A execução da Revisão Sistemática de Literatura foi conduzida a partir da aplicação da expressão geral de busca nas bases científicas selecionadas, seguindo um processo estruturado em quatro etapas principais, conforme o modelo PRISMA: identificação, seleção, elegibilidade e inclusão dos estudos (Moher et al., 2009)**.** Conforme demonstrado na Figura 6.

Figura 6 – Diagrama de sequência das etapas de execução da revisão sistemática



Fonte: Elaborado pelo autor (2026).

4.4.1 Identificação dos Estudos

Na etapa de identificação, a string de busca definida na fase de planejamento foi aplicada nas bases eletrônicas selecionadas. Os resultados obtidos em cada base foram reunidos, permitindo a consolidação inicial do conjunto de estudos recuperados. Nessa etapa, também foi realizada uma adaptação pontual da string de busca para cada base, respeitando as particularidades, como idioma aceito pela base, sem alteração do escopo conceitual da busca.

4.4.2 Seleção dos Estudos

A etapa de seleção teve como objetivo eliminar estudos duplicados e aplicar os critérios iniciais de exclusão. Foram removidos artigos repetidos, e documentos que não se caracterizavam como artigos completos, como resumos, pôsteres ou tutoriais.

Em seguida, procedeu-se à leitura dos títulos e resumos dos estudos restantes, aplicando-se os critérios de inclusão e exclusão estabelecidos no protocolo..

4.4.3 Elegibilidade

Na fase de elegibilidade, os textos completos dos artigos selecionados foram analisados de forma preliminar, com foco em seções como introdução, metodologia e conclusões. Essa leitura teve como objetivo verificar a aderência efetiva do estudo ao escopo da pesquisa e aos critérios definidos.

Os artigos que atenderam a essa avaliação inicial foram, então, submetidos a uma leitura integral e detalhada, permitindo a reaplicação critérios de inclusão e exclusão, bem como a confirmação de sua relevância para os objetivos da revisão.

4.4.4 Inclusão e Extração de Dados

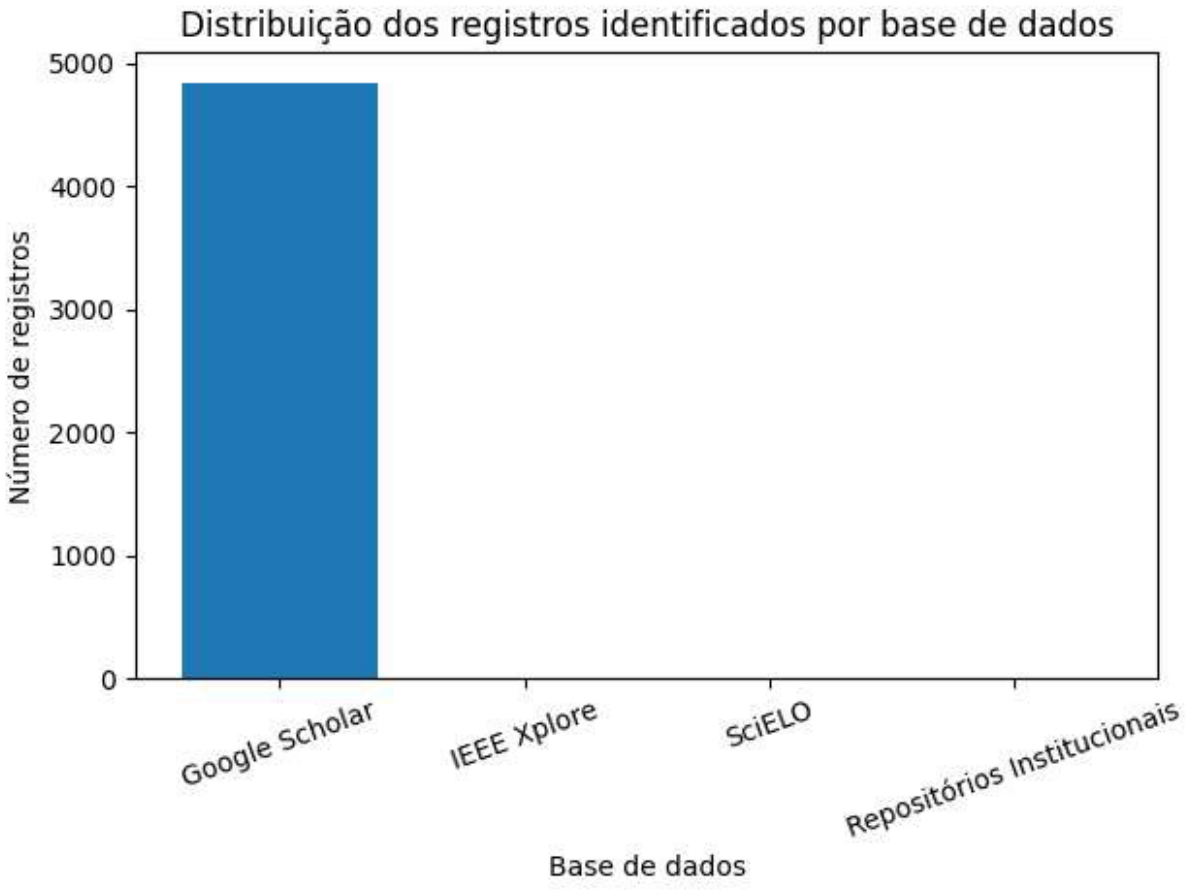
Na etapa final, os estudos considerados elegíveis foram incluídos na revisão sistemática. Quando necessário, informações complementares foram obtidas a partir de fontes associadas aos estudos, como documentações técnicas, repositórios de código ou páginas institucionais, visando melhor compreensão do contexto e das contribuições apresentadas.

Essa etapa permitiu a consolidação dos dados necessários para a síntese dos resultados da revisão, fundamentando a análise comparativa e a discussão dos achados nos capítulos subsequentes.

4.4.5 Seleção de Artigos

As buscas foram realizadas nas bases Google Scholar, IEEE Xplore, SciELO e repositórios institucionais de universidades brasileiras, utilizando a expressão geral de busca definida na Seção 5.3.3. Foram identificados 4.846 registros, sendo 4.840 provenientes do Google Scholar e 6 da IEEE Xplore. Após a remoção de 6 registros duplicados, restaram 4.840 estudos únicos para análise, a Figura 7 mostra a origem dos dados obtidos na busca.

Figura 7 - Distribuição dos registros nas bases de dados



Fonte: Elaborado pelo autor (2026).

Na etapa de identificação, os registros recuperados foram reunidos para análise preliminar. Em seguida, na fase de seleção, os 6 registros provenientes da IEEE Xplore foram excluídos por não atenderem ao escopo do estudo, além de estarem duplicados em relação aos resultados já presentes no Google Scholar. Também foram excluídos estudos de caráter técnico, artigos sem texto completo disponível e publicações fora do domínio de arquitetura de software.

Na etapa de triagem, com base na leitura de títulos e resumos e na aplicação dos critérios de exclusão, 4.820 foram excluídos por não se alinharem diretamente ao escopo da pesquisa, com base na aplicação de critérios de inclusão e exclusão, resumos e palavras-chave, etapa necessária diante da inviabilidade de análise integral de todos os estudos recuperados, resultando em 20 estudos potencialmente relevantes. Esses estudos foram avaliados por meio da leitura do texto completo.

Ao final desse processo, 15 estudos atenderam integralmente aos critérios estabelecidos e foram incluídos na revisão sistemática. Esses estudos compreendem artigos científicos publicados em periódicos e anais de eventos, além de Trabalhos de Conclusão de Curso e dissertações acadêmicas, todos com acesso ao texto completo.

Todos os estudos recuperados foram organizados em uma planilha eletrônica, contendo informações básicas, facilitando a rastreabilidade e a análise posterior[1].

Os estudos incluídos foram então submetidos ao processo de extração e síntese dos dados, conforme o formulário definido (Quadro 3). Nessa etapa, foram identificadas as arquiteturas abordadas, os atributos arquiteturais analisados, os contextos de aplicação e os principais resultados discutidos por cada trabalho.

Os estudos (E1-15) selecionados após a aplicação dos critérios estão apresentados no Quadro 4, que sintetiza as principais características dos trabalhos incluídos nesta revisão sistemática.

Quadro 4 – Detalhamento dos 15 Artigos selecionados.

ID	Título	Autor(es) / Ano	Contexto / Plataforma	Arquiteturas abordadas	Atributos analisados	Objetivo
E1	A comparative study of software architectures in mobile	Dobrean; Diosan (2019)	mobile / geral	MVC/MVVM/VIPER-VIP.	modularização, acoplamento, manutenibilidade	Comparar padrões arquiteturais quanto a desempenh

[1] Disponível em: https://docs.google.com/spreadsheets/d/1dpabqDx3dV1-6nncb0iNyQR0zE-H1T4qnO0laBZf__k/edit?usp=sharing

	applications					o, escalabilidade e manutenção, utilizando métricas experimentais e o método SAAM
E2	Review of ios architectural pattern for testability, modifiability, and performance quality	Sholichin et al. (2019)	iOS	MVC/MVP/MVVM/VIPER	testes/escalabilidade/performance	Analisar padrões arquiteturais sob a perspectiva de qualidade de software
E3	Desenvolvimento de um aplicativo iOS para gestão de eventos com foco em qualidade	Souza (2020)	iOS	MVP, MVC, MVVM, MVVM-C, VIP	qualidade/manutenção/testes	Avaliar o impacto de diferentes arquiteturas iOS no desempenho e na organização do código.
E4	Um estudo comparativo entre padrões arquiteturais para o desenvolvimento de aplicativos para a plataforma iOS	Magalhães(2020)	IOS	MVC, MVVM, VIP	modularização, acoplamento, manutenibilidade	Investigar como a modularização do VIP influencia desempenho e qualidade estrutural em aplicações iOS.
E5	Análise comparativa	Barbosa (2022)	Android	MVC/MVP/MVVM/MVI	modularização, coesão, manutenibilid	Comparar MVVM e VIPER no

	entre os padrões mvc, mvp, mvvm e mvi na plataforma android				ade	contexto do SwiftUI, considerando organização do código e impacto arquitetural.
E6	Arquitetura de software: um estudo orientado ao desenvolvimento de aplicativos móveis híbridos	Pereira; Souza (2023)	mobile/geral	MVC	modularização, coesão, manutenibilidade	Aplicar Clean Architecture, Portas e Adaptadores e MVC em apps híbridos e analisar métricas e testes
E7	Análise comparativa das arquiteturas de software mvc e mvvm no desenvolvimento mobile: desempenho de renderização de tela, consumo de cpu e memória ram	Correia(2025)	Flutter	MVC/MVVM	acoplamento, coesão	Comparar MVC vs MVVM em Flutter medindo tempo de renderização, CPU e RAM, com DevTools e testes de integração.
E8	Modularização de aplicativos iOS	Gomes; Romano; Dias(2023)	iOS	MVC, MVVM	modularização, coesão, testabilidade	Organizar conhecimento para estabelecer arquitetura modular em apps iOS, discutindo desafios de

						implantação conforme o projeto e equipe crescem
E9	Benchmarking common architectural patterns in ios development	Phan (2019)	iOS	MVP/MVVM/VIPER	modularização, acoplamento, testabilidade, acoplamento	Implementar um app iOS em MVC, MVP, MVVM e VIPER e comparar testabilidade, distribuição de responsabilidades, facilidade de desenvolvimento e performance (CPU/RAM)
E10	Detecting model view controller architectural layers using clustering in mobile codebases	Dobrean; Diosan (2020)	mobile/geral	MVC		Detectar camadas arquiteturais (MVC) em codebases mobile e apoiar identificação e eliminação de *architectural smells.*

E11	Automatic examining of software architectures on mobile applications codebases	Dobrean (2019)	mobile/geral	MVP/MVVM/VIPER		Definir método automático para extrair e examinar arquitetura de apps móveis usando SDKs e IA, visando detectar inconsistências cedo
E12	Optimizing architecture in ios development: a comparative study of mvvm and viper using swiftui	Moloudi (2025)	iOS	MVVM/VIPER/VIP		Comparar MVVM vs VIPER em app SwiftUI) por 5 dimensões: esforço, complexidade/manutenibilidade, separação de responsabilidades, testabilidade e performance
E13	Comparison of Architectural Patterns within iOS Applications	Skrypchenko (2024)	iOS	MVC, MVVM, VIPER	modularização, acoplamento, coesão, testabilidade	Comparar arquiteturas iOS (MVC, MVVM, VIPER, propondo um framework para apoiar a escolha arquitetural baseada em objetivos de

						negócio e restrições do projeto
E14	Integração de sistemas de cftv via plataforma de comunicação em nuvem: desenvolvimento de aplicativo móvel	Comelli (2025)	iOS	MVC/MVVM	acoplamento, manutenibilidade	Desenvolver app iOS para integrar CFTV via nuvem, com foco em funcionalidades e segurança
E15	Desenvolvimento do software “stickerz”: um aplicativo para troca de figurinhas da copa do mundo	Silva (2025)	iOS	MVC/MVVM	modularização, coesão, testabilidade	Desenvolver um aplicativo móvel para facilitar a troca de figurinhas da Copa do Mundo, utilizando geolocalização e arquiteturas de software adequadas no frontend e backend.

Fonte: Elaborado pelo autor (2026).

5 Resultados da Revisão Sistemática

Os estudos analisados apresentam discussões e evidências sobre padrões arquiteturais empregados no desenvolvimento de aplicações móveis, com ênfase recorrente em MVC, MVVM e arquiteturas mais segmentadas, como VIPER, e variações como VIP-C, além de abordagens associadas à modularização e à organização do desenvolvimento em equipes. No recorte específico de iOS, a literatura aborda a adoção histórica do MVC e a emergência de

padrões alternativos motivados por limitações percebidas em projetos de maior complexidade, bem como por demandas de evolução contínua (Dobrean; Diosan, 2019; Sholichin et al., 2019; Phan, 2019; Moloudi, 2025; Skrypchenko, 2024).

O conjunto de estudos inclui tanto trabalhos conceituais e descritivos quanto estudos com métricas e comparação experimental. Por exemplo, Sholichin et al. (2019) reportam resultados comparativos entre arquiteturas (MVC, MVP, MVVM e VIPER) considerando testabilidade, modificabilidade, incluindo acoplamento, e desempenho (CPU/memória). Moloudi (2025) apresenta uma comparação empírica entre MVVM e VIPER em SwiftUI, com métricas de complexidade, cobertura de testes e aspectos de desempenho e memória.

Além disso, há estudos que discutem arquitetura como problema de inspeção e validação automatizada, propondo mecanismos para identificar camadas e inconsistências arquiteturais em *codebases* (Dobrean; Diosan, 2020; Dobrean, 2019). Por fim, parte da amostra inclui estudos aplicados ao desenvolvimento de sistemas reais, nos quais MVC e MVVM aparecem como escolhas arquiteturais ou como referência de fundamentação (Comelli, 2025; Silva, 2025). A seguir, as questões de pesquisa da RSL (Quadro 2) são respondidas.

QP1 Quais arquiteturas de software são mais recorrentes na literatura científica e técnica sobre o desenvolvimento de aplicações mobile no ecossistema iOS?

Nos estudos analisados, as arquiteturas MVC e MVVM aparecem como as mais recorrentes na literatura, sendo mencionadas em 12 e 11 dos 15 estudos, respectivamente, o que evidencia seu papel como referências frequentes no ecossistema iOS. O MVC é predominantemente apresentado como ponto de partida arquitetural, enquanto o MVVM surge como alternativa voltada à separação de responsabilidades e à organização do código. Arquiteturas da família VIP, incluindo VIPER e VIP-C, são citadas em 9 estudos, geralmente associadas a discussões sobre modularização, testabilidade e organização estrutural em projetos de maior porte ou com maior complexidade funcional.

A recorrência de MVC, MVVM e VIPER é observada em estudos comparativos e de avaliação que discutem decisões arquiteturais no ecossistema iOS a partir de restrições técnicas, organizacionais e de qualidade (Sholichin et al., 2019; Phan, 2019; Moloudi, 2025; Skrypchenko, 2024). Em especial, Skrypchenko (2024) realiza uma análise comparativa

explícita entre MVC, MVVM e VIPER utilizando aplicações iOS desenvolvidas em Swift, defendendo que a escolha arquitetural deve ser orientada por restrições de projeto e objetivos de negócio. Moloudi (2025), por sua vez, concentra-se na comparação entre MVVM e VIP no contexto do SwiftUI, adotando recorte empírico e métricas estruturais.

De forma pontual, Skrypchenko (2024) amplia o conjunto de arquiteturas analisadas ao incluir a The Composable Architecture (TCA), discutindo modularidade, previsibilidade de estado e estratégias organizacionais. No entanto, a ocorrência dessa arquitetura é mínima no conjunto analisado.

A Figura 8 apresenta uma nuvem de palavras construída a partir da recorrência das arquiteturas mencionadas nos estudos selecionados. Observa-se maior destaque para MVC e MVVM, seguidas por VIPER e variações funcionalmente equivalentes, como VIP-C, enquanto arquiteturas alternativas, como a TCA, aparecem de forma pontual, indicando menor recorrência na literatura analisada.

Figura 8 - Nuvem de palavras das arquiteturas mais recorrentes na literatura iOS

Fonte: Elaborado pelo autor

QP2 Como a literatura descreve e caracteriza arquiteturas MVC, MVVM e VIP-C aplicadas ao desenvolvimento mobile iOS?

Os dados analisados foram sintetizados no Quadro 5 a seguir.

Quadro 5 - Caracterização das arquiteturas MVC, MVVM e VIP-C

Arquitetura	Caracterização segundo a	Principais referências

	literatura	
MVC	Arquitetura tradicional utilizada no desenvolvimento de aplicações iOS, com ênfase no papel central do controlador. A literatura aponta problemas associados ao uso inadequado do padrão e à baixa granularidade dos componentes, indicando que a necessidade de introdução de novas camadas pode emergir conforme a complexidade do sistema aumenta. Também é descrita como arquitetura “default” no ecossistema iOS, associada ao risco do *Massive View Controller* em projetos de maior escala.	Dobrean e Diosan (2019); Sholichin et al. (2019)
MVVM	Arquitetura que desloca parte da lógica de apresentação para o ViewModel e utiliza mecanismos de ligação e gerenciamento de estado para atualização da interface. É apresentada como alternativa ao MVC com o objetivo de aumentar a testabilidade e a modularidade. No contexto do SwiftUI, integra-se de forma mais direta ao paradigma declarativo baseado em estado, embora possa concentrar responsabilidades conforme o crescimento do sistema.	Sholichin et al. (2019); Correia (2025); Moloudi (2025); Skrypchenko (2024)
VIPER / VIP-C	Arquiteturas caracterizadas por forte separação de responsabilidades entre componentes como Interactor, Presenter e	Dobrean (2019); Moloudi (2025); Skrypchenko (2024)

	Router. Embora o termo VIP-C não apareça com frequência como nomenclatura formal padronizada, o papel de coordenação de navegação central é recorrente na literatura. Estudos descrevem objetos dedicados ao gerenciamento de fluxos e transições, distintos dos controladores de tela, sendo o Router tratado como entidade responsável pela navegação, em papel análogo ao Coordinator. Essas arquiteturas são associadas a menor complexidade estrutural por módulo e maior facilidade de isolamento para testes, especialmente em aplicações organizadas em múltiplas funcionalidades.	

Fonte: Elaborado pelo autor (2026)

Os estudos convergem ao descrever a evolução do MVC para arquiteturas mais modulares, como MVVM e VIPER/VIP, como resposta a limitações observadas em projetos de maior porte e complexidade no ecossistema iOS (Dobrean e Diosan 2019; Sholichin et al. 2019; Moloudi 2025; Skrypchenko 2024).

QP3 De que forma as arquiteturas MVC, MVVM e VIP-C impactam atributos de qualidade de software, como modularização, fluxo de dados, acoplamento e coesão, segundo a literatura?

Com base nos estudos analisados, esta seção organiza os principais achados referentes aos impactos das arquiteturas MVC, MVVM e VIP-C sobre atributos de qualidade de software. Sintetizando de forma estruturada, as relações estabelecidas entre essas arquiteturas e seus aspectos . A análise é apresentada no Quadro 6.

Quadro 6 - Atributos de qualidade arquiteturas

Atributo de qualidade	**Síntese dos achados da**	**Principais referências**

	literatura	
Modularização	Gomes, Romano e Dias (2023) discutem a modularização como estratégia de dividir um sistema em subsistemas independentes e flexíveis, vinculando sua adoção ao crescimento do projeto e das equipes. Moloudi (2025) afirma que arquiteturas da família VIP, como o VIPER e suas variações, apresentam maior grau de modularização ao impor uma separação explícita de responsabilidades entre componentes distintos. Em comparação, o MVVM, embora promova separação entre Visão (View) e lógicas por meio da ViewModel, tende a concentrar múltiplas responsabilidades nessa camada, o que pode reduzir a granularidade modular em aplicações maiores.	Gomes, Romano e Dias (2023); Moloudi (2025)
Acoplamento	Sholichin et al. (2019) apresentam uma análise quantitativa baseada em métricas de *data coupling* e *message coupling*, comparando MVC, MVP, MVVM e VIPER. Os resultados indicam que o VIPER apresenta os menores níveis de acoplamento, enquanto o MVVM apresenta valores intermediários e o MVC apresenta maior dependência estrutural. Skrypchenko (2024) associa arquiteturas mais segmentadas a *loose coupling*, permitindo que componentes sejam	Sholichin et al. (2019); Skrypchenko (2024)

	alterados ou substituídos sem afetar significativamente os demais.	
Coesão	Parte dos estudos não mensura coesão diretamente, mas a associa à separação de responsabilidades e ao tamanho e estrutura dos componentes. Sholichin et al. (2019) avaliam a coesão procedural como parte do atributo de modificabilidade e observam melhores resultados para o MVVM, seguido pelo VIPER, com desempenho inferior para o MVC. Moloudi (2025) argumenta que, no VIP/VIPER, a segmentação da arquitetura tende a produzir componentes com responsabilidades mais delimitadas, conceitualmente associadas a maior coesão.	Sholichin et al. (2019); Moloudi (2025)
Fluxo de dados	Moloudi (2025) caracteriza o MVVM como alinhado ao paradigma declarativo, no qual o ViewModel expõe estados observáveis e ações, permitindo atualização automática da interface. Em contrapartida, arquiteturas da família VIP distribuem o fluxo de dados entre múltiplas camadas, tornando explícitas as etapas de transformação ou redirecionamento das informações, o que favorece previsibilidade e testabilidade, ainda que aumente a complexidade estrutural.	Moloudi (2025)

Fonte: Elaborado pelo autor (2026)

Os estudos convergem ao descrever a evolução do MVC para arquiteturas mais modulares, como MVVM e VIPER/VIP, como resposta a limitações observadas em projetos de maior porte e complexidade no ecossistema iOS (Dobrean e Diosan 2019; Sholichin et al. 2019; Moloudi 2025; Skrypchenko 2024).

QP4 Quais evidências a literatura apresenta sobre a relação entre arquiteturas de software e atributos como testabilidade e manutenibilidade em aplicações mobile iOS?

. O Quadro 7 apresenta os resultados a partir dos atributos destacados.

Quadro 7 - Atributos de testabilidade e manutenibilidade

Tipo de evidência	Arquiteturas	Evidências	Referências
Estudos empíricos comparativos baseados em métricas e experimentação	MVC, MVP, MVVM, VIPER	Sholichin et al. (2019) realizam uma comparação quantitativa entre arquiteturas MVC, MVP, MVVM e VIPER, avaliando métricas relacionadas à modificabilidade, testabilidade, acoplamento e consumo de recursos. Os autores identificam que o MVVM apresenta desempenho favorável em testabilidade, enquanto o VIPER se destaca em atributos associados à modificabilidade, incluindo menores níveis de acoplamento e melhor desempenho de CPU. Esses resultados sustentam a associação entre maior segmentação	Sholichin et al. (2019)

		arquitetural e ganhos em atributos que impactam diretamente a manutenibilidade do código.	
Estudos empíricos no contexto iOS	MVVM, VIPER/VIP-C	Moloudi (2025), ao analisar arquiteturas no contexto do desenvolvimento iOS, reporta diferenças observáveis em isolamento de componentes, limites de teste e métricas de complexidade estrutural. O autor associa arquiteturas da família VIP, como o VIPER, a melhor definição de fronteiras para testes unitários e maior modularidade, enquanto caracteriza o MVVM como uma arquitetura mais compacta e de implementação mais rápida, porém sujeita à concentração excessiva de responsabilidades no ViewModel em sistemas de maior porte. Dessa forma, o estudo reforça que a testabilidade está fortemente relacionada à clareza dos limites arquiteturais, embora envolva trade-offs em termos de complexidade organizacional.	Moloudi (2025)

Estudos conceituais e de suporte ferramental	Arquiteturas de software em geral	Outro conjunto de estudos não avalia diretamente arquiteturas por meio de métricas de teste, mas aborda testabilidade e manutenibilidade como consequências do desenvolvimento arquitetural ao longo do tempo. Dobrean (2019) e Dobrean; Diosan (2020) investigam métodos automáticos para detecção de camadas arquiteturais, dependências indevidas e inconsistências estruturais em aplicações mobile. Os autores argumentam que problemas arquiteturais afetam negativamente atributos internos do software, como manutenibilidade e extensibilidade, ao promoverem degradação da estrutura. Nesses trabalhos, a relação entre arquitetura e testabilidade é indireta: arquiteturas mais bem definidas e monitoradas tendem a sofrer menos degradação, o que reduz o custo de manutenção e facilita intervenções futuras. Assim, a arquitetura é tratada como um	Dobrean (2019); Dobrean; Diosan (2020)

		artefato que deve ser continuamente validado, e não apenas como uma decisão tomada na fase inicial do projeto.	

Fonte: Elaborado pelo autor (2026).

Os estudos indicam que arquiteturas mais segmentadas, como MVVM e VIP/VIPER, apresentam potenciais ganhos em testabilidade e manutenibilidade, especialmente quando avaliadas por métricas quantitativas. Estudos de natureza conceitual e ferramental reforçam essa relação ao demonstrar que a preservação da arquitetura ao longo do ciclo de vida reduz problemas de manutenção. No entanto, os trabalhos também evidenciam que tais benefícios dependem da correta aplicação da arquitetura e do contexto de escala e complexidade do projeto, não sendo inerentes unicamente à adoção de um padrão arquitetural.

QP5 Quais benefícios e desafios são relatados na literatura quanto à adoção das arquiteturas MVC, MVVM e VIP-C em projetos iOS?

O Quadro 8 apresenta os resultados a partir dos atributos destacados.

Quadro 8 - Benefícios e desafios das arquiteturas

Arquitetura	Benefícios	Desafios	Principais referências
MVC	Arquitetura de baixa complexidade inicial e fácil adoção. Apresenta simplicidade inicial e praticidade no desenvolvimento, sendo amplamente utilizada no ecossistema iOS.	Associada a dificuldades crescentes de teste e manutenção em projetos de maior escala, especialmente em razão da concentração excessiva de responsabilidades no Controller, fenômeno descrito como *Massive View Controller*.	Sholichin et al. (2019); Comelli (2025)

MVVM	Favorece desenvolvimento mais rápido e direto, sobretudo em aplicações SwiftUI de pequeno e médio porte, devido à integração com mecanismos de estado e atualização declarativa da interface. Associada a melhores condições de testabilidade e modificabilidade quando comparada ao MVC. Benefícios como separação de responsabilidades e facilidade de manutenção são destacados em estudos aplicados.	Impõe maior custo estrutural e pode apresentar concentração excessiva de responsabilidades no ViewModel em sistemas de maior porte. Envolve maior custo de aprendizado quando comparada ao MVC. Evidências aplicadas não fornecem métricas quantitativas de testabilidade ou manutenibilidade.	Sholichin et al. (2019); Moloudi (2025); Silva (2025)
VIP-C / VIPER	Associada a benefícios mais claros quando se priorizam modularidade, testabilidade e manutenção a longo prazo. A separação explícita de responsabilidades e o uso de entidades dedicadas à organização de fluxos de navegação contribuem para melhor organização estrutural e redução de problemas de manutenção.	Impõe maior sobrecarga arquitetural, maior custo estrutural e maior exigência de conhecimento técnico. Envolve desafios associados à implementação e validação ao longo do ciclo de vida do software.	Sholichin et al. (2019); Moloudi (2025); Dobrean (2019)

Fonte: Elaborado pelo autor (2026).

Em resumo, a literatura aponta que o MVC se destaca pela simplicidade inicial, mas

apresenta limitações relevantes de testabilidade e manutenção em projetos de maior escala. O MVVM é caracterizado como uma solução intermediária, oferecendo melhor separação de responsabilidades e ganhos estruturais em contextos de pequeno a médio porte. As arquiteturas da família VIP, incluindo o VIP-C, são associadas a melhores condições de modularidade, testabilidade e sustentabilidade do código em sistemas mais complexos, embora impliquem maior complexidade estrutural. Os estudos convergem ao indicar que esses benefícios e desafios dependem do contexto de aplicação e da forma como a arquitetura é efetivamente empregada.

QP6 Em quais contextos de projeto (aplicações iniciais, sistemas em evolução ou projetos de maior porte) a literatura indica maior adequação das arquiteturas MVC, MVVM e VIP-C?

A análise indica que a adequação das arquiteturas MVC, MVVM e VIP-C está principalmente associada a três fatores contextuais: porte do sistema, horizonte de evolução e restrições do projeto. O porte do sistema é discutido em 10 dos 15 estudos, o horizonte de evolução, relevante em 8 estudos, e restrições técnicas e organizacionais como tamanho da equipe, exigência de cobertura de testes e complexidade dos fluxos, considerados em 5 estudos (Sholichin et al., 2019; Dobrean; Diosan, 2019; Skrypchenko, 2024; Moloudi, 2025).

No contexto de aplicações iniciais ou com pequeno escopo, 3 estudos indicam maior adequação de arquiteturas menos segmentadas, como MVC ou MVVM, especialmente em cenários caracterizados por baixa complexidade funcional, poucos fluxos de navegação e menor exigência de isolamento estrutural (Sholichin et al., 2019; Moloudi, 2025; Skrypchenko, 2024). Moloudi (2025) destaca que, nesses casos, a MVVM favorece entregas rápidas e maior produtividade, sobretudo em aplicações pequenas e em equipes reduzidas, enquanto Dobrean (2019) argumenta que o MVC pode ser suficiente quando a complexidade é contida, não sendo justificável a adição de estruturas de coordenação.

Além disso, 3 trabalhos associam projetos médios ou de maior porte, bem como sistemas com longa expectativa de evolução, a limitações do MVC, destacando dificuldades crescentes relacionadas à testabilidade, manutenção e organização do código. Nesses contextos, arquiteturas mais segmentadas, como VIPER ou VIP-C, são apontadas como mais

adequadas para preservar atributos de qualidade ao longo do ciclo de vida do software (Sholichin et al., 2019; Dobrean; Diosan, 2019; Moloudi, 2025).

Adicionalmente, 2 estudos enfatizam que restrições específicas do projeto, como tamanho da equipe e complexidade dos fluxos de navegação, influenciam diretamente a escolha arquitetural, indicando que arquiteturas com separação explícita de responsabilidades e mecanismos dedicados de coordenação de fluxos tornam-se mais apropriadas nesses cenários (DOBREAN, 2019; SKRYPCHENKO, 2024). A VIP-C aparece como relevante em aplicações com múltiplos fluxos e navegação complexa, nas quais a separação da lógica de navegação dos controladores de interface contribui para melhor organização estrutural (DOBREAN, 2019).

Por fim, estudos aplicados reforçam essas conclusões ao descreverem o uso de MVC e MVVM em sistemas reais com requisitos de robustez, integração e segurança, ainda que a validação nesses casos seja realizada por testes manuais e não constitua evidência empírica comparativa (COMELLI, 2025).

5.1 Síntese da Revisão Sistemática da Literatura

De forma conclusiva, a literatura indica que o MVC apresenta maior adequação em aplicações iniciais ou de escopo reduzido, nas quais a simplicidade estrutural, os baixos requisitos do projeto e a agilidade de entrega são prioritárias. Contudo, os estudos também evidenciam que, à medida que a complexidade cresce, o MVC tende a demandar refinamentos estruturais ou a incorporação de outros padrões para mitigar limitações relacionadas à testabilidade, modularização e manutenção, o que reduz sua adequação em projetos de médio e grande porte.

O MVVM, por sua vez, é apontado como uma alternativa intermediária, particularmente adequada a aplicações de pequeno a médio porte, sobretudo no contexto do SwiftUI, em que o paradigma declarativo favorece a separação entre interface e lógica de apresentação. Conclui-se que o MVVM oferece melhor equilíbrio entre simplicidade e organização estrutural quando comparado ao MVC, embora possa apresentar desafios de escalabilidade caso haja concentração excessiva de responsabilidades no ViewModel nos sistemas em evolução.

Já as arquiteturas da família VIP, incluindo variações como o VIP-C e VIPER, são consistentemente associadas a projetos de maior porte, longa duração ou elevada complexidade, nos quais atributos como testabilidade, modularização e previsibilidade estrutural são prioritários. A separação explícita de responsabilidades e a introdução de entidades dedicadas à coordenação de fluxos tornam essas arquiteturas mais adequadas a cenários com múltiplos fluxos de navegação e maior necessidade de isolamento de componentes, ainda que à custa de maior complexidade organizacional.

De acordo com a revisão sistemática, recomenda-se a adoção do MVC em aplicações iniciais ou de escopo reduzido, nas quais a complexidade funcional é limitada e os requisitos de evolução são restritos. O MVVM mostra-se mais adequado em projetos de pequeno a médio porte, especialmente no ecossistema SwiftUI, quando se busca maior organização estrutural sem a introdução de uma segmentação arquitetural excessiva. Já as arquiteturas da família VIP, incluindo variações como o VIP-C/VIPER, são indicadas em projetos de maior porte, longa duração ou elevada complexidade funcional, nos quais a separação rigorosa de responsabilidades, a previsibilidade estrutural e a testabilidade assumem papel central. Assim, as diretrizes extraídas desta revisão sistemática estabelecem que a escolha arquitetural é resultado da correspondência entre o escopo inicial, a expectativa de evolução, a complexidade do domínio e os atributos de qualidade priorizados, reforçando a arquitetura como uma decisão estratégica no desenvolvimento de aplicações iOS.

As diretrizes podem ser apresentadas de forma resumida de acordo com o Quadro 9.

Quadro 9 - Diretrizes de escolha arquitetural no desenvolvimento de aplicações iOS

Arquitetura	**Contexto de projeto indicado**	**Porte típico do projeto**	**Situações de uso recomendadas**	**Principais vantagens associadas**
MVC	Aplicações iniciais ou de escopo reduzido	Pequeno	Projetos com baixa complexidade funcional, poucos fluxos de navegação e requisitos de evolução restritos	Simplicidade estrutural, facilidade de adoção e rapidez no desenvolvimento

MVVM	Projetos de pequeno a médio porte, especialmente no SwiftUI	Pequeno a médio	Quando se busca maior organização estrutural e melhor separação entre interface e lógica de apresentação, sem segmentação arquitetural excessiva	Melhor da estabilidade e organização estrutural em relação ao MVC, integração natural com paradigma declarativa
VIP / VIP-C / VIPER	Projetos de maior porte, longa duração ou elevada complexidade funcional	Grande	Sistemas com múltiplos fluxos de navegação, alta exigência de previsibilidade estrutural, isolamento de responsabilidades e testabilidade	Alta modularidade separação rigorosa de responsabilidades maior previsibilidade em escala

Fonte: Elaborado pelo autor (2026)

5.2 Considerações Finais da Síntese

De forma consolidada, a revisão conduzida ao longo deste capítulo indica que a escolha arquitetural no ecossistema iOS não pode ser tratada como uma decisão neutra ou puramente convencional, devendo considerar explicitamente o contexto do projeto, o porte do sistema, suas possibilidades de evolução e os atributos de qualidade priorizados. A análise demonstra que diferentes arquiteturas impõem estruturas distintas de organização e separação de responsabilidades, as quais impactam diretamente aspectos como modularização, testabilidade, manutenibilidade , coesão e organização do fluxo de dados.

Os estudos revisados sustentam que arquiteturas mais segmentadas, como MVVM e VIP-C, são mais adequadas em cenários nos quais a previsibilidade estrutural, a separação clara de responsabilidades e a sustentabilidade do código ao longo do tempo são requisitos centrais. Em contrapartida, arquiteturas menos segmentadas, como o MVC, mostram-se

adequadas em contextos de menor complexidade, nos quais a estrutura do sistema permanece contida e as demandas de evolução são limitadas, ainda que apresentem restrições quando aplicadas a sistemas em crescimento.

Assim, a principal contribuição desta síntese não resultou na identificação de uma arquitetura superior de forma absoluta, mas na derivação de diretrizes de escolha arquitetural fundamentada na comparação entre MVC, MVVM e VIP-C.

6 INVESTIGAÇÃO COM O PÚBLICO-ALVO

Este capítulo apresenta uma investigação realizada por meio de um questionário direcionado a desenvolvedores iOS, com a finalidade de complementar os achados da Revisão Sistemática da Literatura. As respostas na íntegra de cada voluntário estão disponíveis em repositório no google docs[2], que reúne os dados brutos e os gráficos correspondentes, a análise aqui desenvolvida busca articular as percepções empíricas dos participantes com os resultados discutidos na literatura, permitindo uma abordagem interpretativa.

A investigação foi conduzida por meio de um formulário estruturado, elaborado e disponibilizado na plataforma Google Forms, durante um período de 7 dias corridos. A divulgação foi feita por meio de fóruns de desenvolvedores mobile iOS e no canal oficial de comunicação dos alunos do Curso de Sistemas e Mídias Digitais da Universidade Federal do Ceará.

O instrumento de coleta foi composto por 17 questões, destinadas a caracterizar o perfil profissional e acadêmico dos respondentes, bem como a levantar informações sobre sua experiência prática e histórico de utilização de arquiteturas de software no desenvolvimento de aplicações iOS. As questões foram formuladas de modo a permitir a comparação entre as percepções empíricas dos participantes e os resultados obtidos na análise da Revisão Sistemática da Literatura.

A versão integral do questionário aplicado encontra-se disponível no Apêndice A, possibilitando a consulta às questões investigadas.

[2]Disponível em: https://docs.google.com/spreadsheets/d/1hNV_hfOen-B57aJDCa8zMlS4supAG2JB-9WUoEJ_9Eg/edit?usp=sharing

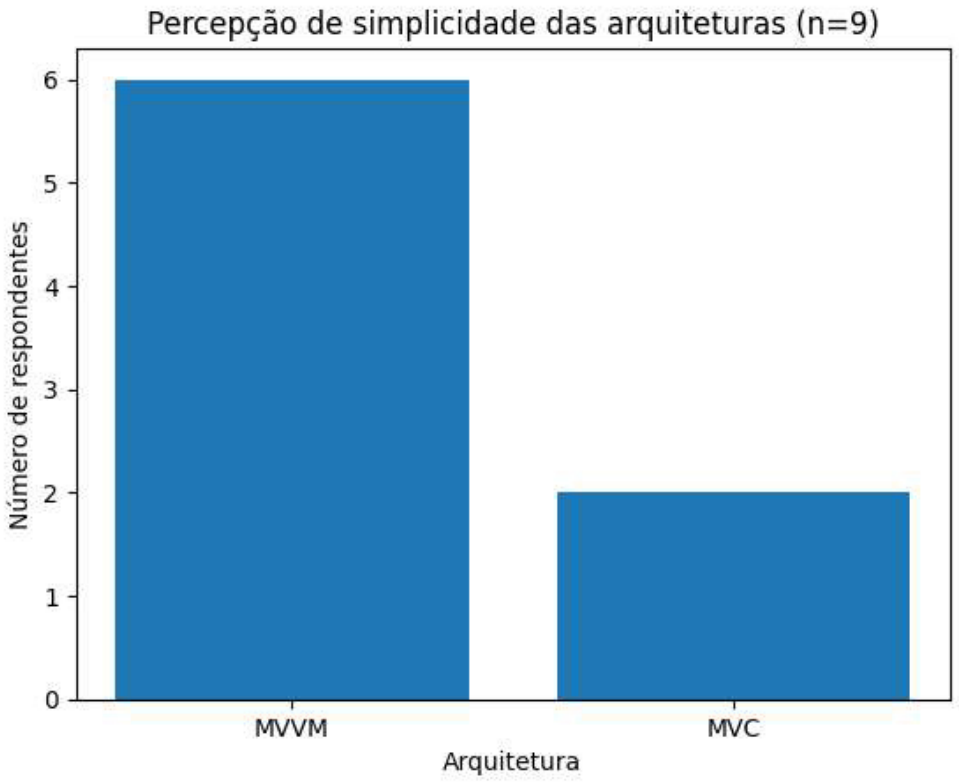
A amostra do questionário foi composta por 9 respondentes, todos com experiência prática no desenvolvimento de aplicações iOS. Desse total, 7 participantes relataram uso recorrente de MVC e MVVM, enquanto 4 mencionaram explicitamente arquiteturas, como VIPER/VIP-C. Esse dado empírico indica a predominância de MVC e MVVM como arquiteturas recorrentes no ecossistema iOS, ao mesmo tempo em que evidencia a adoção seletiva de arquiteturas mais estruturadas em contextos específicos, conforme descrito na literatura revisada.

No que se refere ao momento da escolha arquitetural, 6 dos 9 respondentes indicaram que a decisão ocorreu antes do início do desenvolvimento, enquanto 2 relataram que a arquitetura foi definida ou revisada durante a implementação, em resposta a problemas estruturais emergentes. Esse padrão empírico reforça a constatação presente na literatura de que limitações arquiteturais tendem a se manifestar durante a evolução do sistema, levando à adoção tardia de soluções mais segmentadas quando a complexidade já está instalada.

A análise das motivações para a escolha arquitetural evidencia que padrões previamente adotados pela equipe ou organização e preocupações com manutenção e evolução futura do sistema foram os fatores mais recorrentes, superando análise de critérios técnicos . Esse achado empírico dialoga diretamente com os estudos analisados na RSL, que apontam a arquitetura como uma decisão fortemente influenciada por fatores organizacionais e históricos, e não apenas por avaliação técnica formal de uma arquitetura adequada ao contexto da aplicação.

Quanto à percepção de simplicidade, 6 dos respondentes indicaram o MVVM como a arquitetura mais simples de utilizar, enquanto apenas 2 apontaram o MVC, o que a priore, contraria a associação tradicional do MVC como abordagem mais simples, conforme apresentado na Figura 9. Esse resultado pode indicar, que no contexto atual do desenvolvimento iOS, especialmente com a adoção do SwiftUI, o MVVM é percebido como estruturalmente mais compreensível e alinhado ao fluxo de desenvolvimento mais moderno.

Figura 9 – Percepção de simplicidade entre as arquiteturas MVVM e MVC



Fonte: Elaborado pelo autor (2026).

Os dados em escala Likert apresentam elevado grau de convergência. Em praticamente todos os itens relacionados à separação de responsabilidades, modularização, clareza do fluxo de dados, testabilidade e manutenibilidade, mais de 80% dos respondentes marcaram “Concordo totalmente”, com o restante concentrado em “Concordo parcialmente”. Não foram registradas discordâncias totais. Em especial, todos os respondentes válidos concordaram totalmente que:

- a separação clara entre lógica de negócio, apresentação e interface facilita a escrita de testes unitários;
- decisões arquiteturais iniciais influenciam diretamente o esforço de manutenção e modificação ao longo do tempo.

Esses resultados empíricos fornecem suporte direto às conclusões da Revisão Sistemática da Literatura, a qual indica que arquiteturas mais segmentadas tendem a oferecer melhores condições para testabilidade e manutenção, especialmente em sistemas de maior porte ou em evolução contínua, sem caracterizar tais benefícios como universais ou

automáticos. A convergência entre os achados empíricos e os resultados da literatura reforça que os impactos positivos dessas arquiteturas estão condicionados ao contexto do projeto, ao domínio da aplicação e ao grau de maturidade da equipe de desenvolvimento.

As respostas discursivas reforçam e qualificam a análise quantitativa. Os relatos mencionam, de forma recorrente, a concentração excessiva de responsabilidades em classes centrais, a ausência de uma definição arquitetural explícita, a necessidade de refatorações tardias e a dificuldade de evolução do código como problemas enfrentados nos projetos dos quais os participantes fizeram parte. Esses aspectos dialogam diretamente com os padrões problemáticos descritos na literatura, como a sobrecarga estrutural associada ao uso prolongado do MVC em sistemas em crescimento, frequentemente materializada na forma de *Massive View Controllers*. Em múltiplos casos, os participantes relatam que tais dificuldades motivaram a adoção posterior de arquiteturas mais estruturadas ou de padrões adicionais, o que está em consonância com os estudos que descrevem a introdução tardia de camadas arquiteturais como uma resposta corretiva a decisões iniciais insuficientemente planejadas.

De forma integrada, os dados do questionário não apenas confirmam, mas operacionalizam empiricamente os principais argumentos levantados pela revisão sistemática, ao demonstrar que os trade-offs estruturais discutidos nos estudos científicos são reconhecidos e vivenciados na prática profissional. Assim, a investigação empírica cumpre seu papel como evidência descritiva complementar, fortalecendo a validade das diretrizes de escolha arquitetural propostas neste trabalho e explicitando a relação entre decisões arquiteturais, atributos de qualidade e contextos reais de desenvolvimento no ecossistema iOS.

6.1 Limitações Da Análise

Esta análise apresenta limitações que devem ser consideradas na interpretação de seus resultados. A principal limitação da análise empírica refere-se ao número reduzido de respondentes (n = 9), o que impede qualquer generalização estatística dos achados. Os dados coletados representam percepções individuais e contextos específicos de atuação profissional, devendo ser interpretados apenas como evidência descritiva e ilustrativa que indicam a direção dos achados na RSL, mas necessitam de uma amostra maior para serem validados de forma generalizada.

Além disso, as respostas possuem caráter autodeclaratório, estando sujeitas a vieses de percepção pessoal, memória e experiência prévia dos participantes voluntários. O questionário não teve como objetivo mensurar quantitativamente atributos de qualidade de software, mas identificar a correlação padrões de resposta com os resultados obtidos na análise comparativa.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve como objetivo analisar comparativamente as arquiteturas MVC, MVVM e VIP-C no contexto do desenvolvimento de aplicações mobile iOS, com ênfase em atributos de qualidade de software, tais como modularização, acoplamento e coesão, testabilidade, manutenibilidade e adequação a diferentes contextos de projeto. Para isso, foi conduzida uma revisão sistemática e analítica da literatura, permitindo uma visão abrangente sobre os impactos arquiteturais discutidos na área.

Os resultados obtidos indicam que não existe uma arquitetura universalmente superior, mas sim abordagens com características distintas, cujas adequações variam conforme o porte do sistema, sua necessidade de evolução e os requisitos de qualidade priorizados.

A arquitetura MVC mostrou-se adequada em aplicações iniciais e de escopo reduzido, nas quais a simplicidade estrutural é suficiente para atender às demandas do projeto. Entretanto, a literatura evidencia limitações dessa abordagem quando aplicada a sistemas em crescimento, sobretudo no que se refere à manutenção da separação de responsabilidades e à testabilidade.

O MVVM foi identificado como uma alternativa intermediária, oferecendo melhor organização estrutural em comparação ao MVC e favorecendo aplicações de pequeno a médio porte. Ainda assim, os estudos indicam que, em projetos em evolução, o MVVM pode apresentar desafios relacionados à concentração de responsabilidades no ViewModel.

Além disso, observou-se que arquiteturas da família VIP, incluindo variações como o VIP-C/VIPER, mostraram-se mais adequadas a projetos de maior porte, longa duração ou elevada complexidade funcional.

Como principal contribuição, este trabalho apresenta diretrizes de escolha arquitetural fundamentada nos estudos apresentados, capazes de auxiliar desenvolvedores e equipes iOS a selecionar a arquitetura mais adequada para diferentes cenários de projeto. Essa diretriz

reforça a arquitetura como uma decisão estratégica de Engenharia de Software, que deve ser tomada de forma consciente e contextualizada, e não apenas com base em convenções ou preferências técnicas.

Como trabalhos futuros e desdobramentos desta pesquisa, existem as seguintes possibilidades:

- realização de estudos empíricos controlados comparando implementações reais das arquiteturas MVC, MVVM e VIP-C em aplicações iOS equivalentes;
- investigação quantitativa mais aprofundada do impacto arquitetural sobre métricas de desempenho, consumo de recursos e cobertura de testes;
- ampliação da análise para arquiteturas emergentes no ecossistema iOS, incluindo abordagens híbridas;
- estudos voltados à aplicação dessas arquiteturas em contextos educacionais, projetos de pesquisa, analisando sua influência no processo de aprendizagem e na organização de projetos acadêmicos.
- ampliar a investigação com o público-alvo;
- investigações em empresas.

REFERÊNCIAS

APPLE INC. *The Swift Programming Language*. San Francisco: Apple Inc., 2014. Disponível em: https://books.apple.com/us/book/the-swift-programming-language-swift-5-1/id881256329. Acesso em: 10 dez. 2025.

APPLE INC. *Apple unveils groundbreaking new technologies for app development: Breakthrough SwiftUI Framework, ARKit 3 and New Xcode Tools make developing powerful apps easier and faster than ever*. San Jose, CA: Apple Inc., 3 jun. 2019. Disponível em: https://www.apple.com/newsroom/2019/06/apple-unveils-groundbreaking-new-technologies-for-app-development/. Acesso em: 15 dez. 2025.

APPLE INC. *Model-View-Controller*. Cupertino, CA: Apple Inc., 06 abr. 2018. Disponível em: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html. Acesso em: 15 nov. 2025.

APPLE. iOS Adoption Data. 2023. Disponível em: https://developer.apple.com/support/app-store/. Acesso em: 15 nov. 2025.

APPLE. iOS Developer Documentation. 2023. Disponível em: https://developer.apple.com/documentation/. Acesso em: 10 nov. 2025.

APPLE. Introduction to MVVM in iOS. Apple Developer Documentation, 2023. Disponível em: https://developer.apple.com/documentation. Acesso em: 14 nov. 2025.

APPLE INC. *SwiftUI*. Cupertino, CA: Apple Inc., 2025. Disponível em: https://developer.apple.com/documentation/SwiftUI. Acesso em: 14 dez. 2025.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 3. ed. Boston: Addison-Wesley, 2013.

BARBOSA, Adriley Samuel Ribeiro. Análise comparativa entre os padrões MVC, MVP, MVVM e MVI na plataforma Android. 2022. Trabalho de Conclusão de Curso (Bacharelado

em Sistemas de Informação) – Instituto Federal Goiano, Campus Urutaí, Urutaí, 2022. Disponível em: https://repositorio.ifgoiano.edu.br/handle/prefix/2611. Acesso em: 20 nov. 2025.

BRERETON, P.; KITCHENHAM, B.; BUDGEN, D.; TURNER, M.; KHALIL, M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, v. 80, n. 4, p. 571–583, 2007.

CARDOSO, Gustavo. Clean Architecture e MVVM no desenvolvimento Android. *Medium*, 8 jun. 2021. Disponível em: https://medium.com/@cardosof.gui/clean-architecture-e-mvvm-no-desenvolvimento-android-6f542d0f2e99. Acesso em: 08 jan. 2026.

CLEAN SWIFT. Clean Swift iOS Architecture for Fixing Massive View Controller. 2019. Disponível em: https://clean-swift.com/clean-swift-ios-architecture/. Acesso em: 20 nov. 2025.

COMELLI, Cesar Henrique Campos. Integração de sistemas de CFTV via plataforma de comunicação em nuvem: desenvolvimento de aplicativo móvel. 2025. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) — Universidade Federal de Santa Catarina, Centro Tecnológico, Florianópolis, SC, 26 fev. 2025. Disponível em: https://repositorio.ufsc.br/handle/123456789/264398. Acesso em: 12 jan. 2026.

CORREIA, Adson Bruno de Souza. Análise comparativa das arquiteturas de software MVC e MVVM no desenvolvimento mobile: desempenho de renderização de tela, consumo de CPU e memória RAM. 2025. 38 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Alagoas, Campus Arapiraca, Arapiraca, 26 maio 2025. Disponível em: https://ud10.arapiraca.ufal.br/repositorio/publicacoes/5957. Acesso em: 12 jan. 2026.

CURSA. Arquiteturas iOS: MVC, MVVM e VIPER. Disponível em: https://www.cursa.com.br/curso/ios-arquiteturas. Acesso em: 15 out. 2025.

DATA.AI. *State of Mobile 2023*. 2023. Disponível em: https://cocktailmarketing.com.mx/wp-content/uploads/2023/12/State-of-mobile-2023-data.ai_.pdf. Acesso em: 14 nov. 2025.

DOBREAN, Dragoș. Automatic examining of software architectures on mobile applications codebases. In: *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) — Doctoral Symposium*, IEEE, p. 595-599, 2019. Disponível em: https://dobrean.ro/wp-content/uploads/2019/10/272-automatic-examining-of-software-architectures-on-mobile-applications-codebases-1.pdf. Acesso em: 12 jan. 2026.

DOBREAN, Dragoș; DIOȘAN, Laura. Detecting Model View Controller Architectural Layers using Clustering in Mobile Codebases. In: *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, SciTePress – Science and Technology Publications, Cluj-Napoca, Romania, p. 196-203, 2020. DOI: 10.5220/0009884601960203. Disponível em: https://www.scitepress.org/Papers/2020/98846/98846.pdf. Acesso em: 12 jan. 2026.

DOBREAN, Dragos; DIOȘAN, Laura. Model View Controller in iOS mobile applications development. In: *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering (SEKE 2019)*, Cluj-Napoca, Romania, 2019. DOI: 10.18293/SEKE2019-048. Disponível em: https://ksiresearch.org/seke/seke19paper/seke19paper_48.pdf. Acesso em: 12 jan. 2026.

DRESCH, A.; LACERDA, D. P.; ANTUNES JR., J. A. V. *Design Science Research*. Porto Alegre: Bookman, 2015.

FOWLER, M.Refactoring: improving the design of existing code. Reading, MA: Addison-Wesley, 1999.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2002.

GARCIA, Joshua; POPESCU, Daniel; EDWARDS, George; MEDVIDOVIC, Nenad. *Toward a catalogue of architectural bad smells*. In: *Architectures for Adaptive Software Systems (QoSA)*, 2009. Springer International Publishing. DOI: 10.1007/978-3-642-02351-4_10. Disponível em: https://jgarcia.ics.uci.edu/wp-content/uploads/10.1.1.183.9958.pdf. Acesso em: 15 jan. 2026.

GEEKSFORGEEKS. MVC Architecture – System Design. Disponível em: https://www.geeksforgeeks.org/system-design/mvc-architecture-system-design/. Acesso em: 6 jan. 2026.

GOMES, A. S. Análise de arquiteturas móveis no desenvolvimento de aplicações iOS. *Revista da FATEC Praia Grande*, v. ?, n. ?, p. ?, 2023. Disponível em: https://revistas.fatecpg.edu.br. Acesso em: 16 out. 2025.

GOMES, Matheus Francisco da Silva Lima; ROMANO, Simone Maria Viana; DIAS, Jonatas Cerqueira. Modularização de aplicativos iOS / Modularization of iOS applications. *Revista Processando o Saber*, v. 15, n. 01, p. 01–15, 6 jun. 2023. DOI: 10.5281/zenodo.14927759. Disponível em: https://www.fatecpg.edu.br/revista/index.php/ps/article/view/270. Acesso em: 12 jan. 2026.

GOSSMAN, John. Introduction to Model/View/ViewModel pattern for building WPF apps. Microsoft Developer Network, 2005. Disponível em: https://learn.microsoft.com/en-us/archive/b

logs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps. Acesso em: 12 jan. 2026.

HALAI, Eeshwar. Breaking Free from MVC: How MVVM Transformed Our iOS Development at Eventogy. *Eventogy Blog*, 24 ago. 2024. Disponível em: https://www.eventogy.com/blog/breaking-free-from-mvc-how-mvvm-transformed-our-ios-development-at-eventogy. Acesso em: 8 jan. 2026.

HOSSAIN, Ahmad Shabibul. Breaking Down Massive View Controller Using SOLID & MVP. *Code With Shabib*, 2021. Disponível em: https://www.codewithshabib.com/better-architecture-in-ios-breaking-down-massive-view-controller-using-solid-mvp-for-dummies/. Acesso em: 19 nov. 2025.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

Geneva: ISO, 2011.

KHANLOU, Soroush. The Coordinator. 2015. Disponível em: https://khanlou.com/2015/10/coordinators-redux/. Acesso em: 11 jan. 2026.

KITCHENHAM, B.; PRETORIUS, R.; BUDGEN, D.; BRERETON, P.; TURNER, M.; NIAZI, M.; LINKMAN, S. Systematic literature reviews in software engineering – A tertiary study. *Information and Software Technology*, v. 52, n. 8, p. 792–805, 2009.

LAVOIE, Raymond. Clean Swift iOS Architecture for Fixing Massive View Controller. *Clean Swift*, 2015. Disponível em: https://clean-swift.com/clean-swift-ios-architecture/. Acesso em: 04 nov. 2025.

LAW, Raymond. Coordinators + VIP. Disponível em: https://www.raywenderlich.com/158-coordinator-pattern-tutorial-for-ios. Acesso em: 8 jan. 2026.

LAW, Raymond. *Coordinators Redux*. 2015. Disponível em:

https://khanlou.com/2015/10/coordinators-redux/.

Acesso em: 15 dez 2025.

MAGALHÃES, Ícaro Lima. Um estudo comparativo entre padrões arquiteturais para o desenvolvimento de aplicativos para a plataforma iOS. 2018. Monografia (Trabalho de Conclusão de Curso) — Centro de Informática, Universidade Federal da Paraíba, João Pessoa, 2018. Disponível em: https://repositorio.ufpb.br/jspui/handle/123456789/15651. Acesso em: 16 jan. 2026.

MARTIN, Robert C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston: Prentice Hall, 2018.

MICIANO, Fábio. Arquitetura iOS VIP. *Medium*, 2021. Disponível em: https://medium.com/@fabio.miciano/arquitetura-ios-vip-4b95f9a0faaa. Acesso em: 8 jan. 2026.

MICROSOFT. Architectural patterns: Model-View-ViewModel (MVVM). Microsoft Learn, 2023. Disponível em:

https://learn.microsoft.com/en-us/windows/communitytoolkit/mvvm/introduction. Acesso em: 11 jan. 2026.

MICROSOFT. *Model-View-ViewModel (MVVM)*. Microsoft Learn, 2024. Disponível em: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. Acesso em: 11 jan. 2026.

MOHER, D. et al. Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. *PLoS Medicine*, v. 6, n. 7, e1000097, 2009.

MOULOUDI, Behzad. Optimizing architecture in iOS development: a comparative study of MVVM and VIPER using SwiftUI. 2025. 86 f. Master's Thesis (Master of Engineering in Interactive Technologies) - Turku University of Applied Sciences, Turku, Finland, 2025. Disponível em: https://www.theseus.fi/handle/10024/908051. Acesso em: 12 jan. 2026.

NUBANK. Vaga para Desenvolvedor iOS. Disponível em: https://boards.greenhouse.io/nubank. Acesso em: 16 out. 2025.

NETO, M. V. D. Software Architecture and Agile Development: Perceptions on Quality and Scalability. In: SBCARS – Brazilian Symposium on Software Architecture, 2025.

NORMANDO, Célio. Arquitetura MVC e princípios de projeto. Medium, 1 mar. 2024. Disponível em: https://medium.com/@celionormando/arquitetura-mvc-e-princ%C3%ADpios-de-projeto-3d0b278ef910. Acesso em: 12 jan. 2026.

PHAN, Dang Hai. Benchmarking common architectural patterns in iOS development. 2019. 60 f. Trabalho de Conclusão de Curso (Bachelor of Engineering in Information Technology) - Metropolia University of Applied Sciences, Helsinki, Finlândia, 04 abr. 2019. Disponível em: https://www.theseus.fi/bitstream/10024/167317/2/Hai_Phan.pdf. Acesso em: 12 jan. 2026.

PEREIRA, Gabriel Davi Silva; SOUZA, Danillo Gonçalves de. Arquitetura de software: um estudo orientado ao desenvolvimento de aplicativos móveis híbridos. 2023. 130 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) — Universidade de Brasília,

Brasília, DF, Brasil, 17 fev. 2023. Disponível em: https://bdm.unb.br/handle/10483/35946. Acesso em: 12 jan. 2026.

PEREIRA, Souza, Gonçalves. Arquitetura de software: um estudo orientado ao desenvolvimento mobile. 2023. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) – Universidade de Brasília, Brasília, 2023. Disponível em: https://bdm.unb.br/handle/10483/32219. Acesso em: 20 nov. 2025.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2020.

REENSKAUG, Trygve. Models–Views–Controllers. Xerox PARC, 1979. Documento técnico original do MVC.

RIBEIRO, Caio. MVVM: Entendendo o padrão arquitetural. *Swift by Sundell (Brasil)*, 2021. Disponível em: https://www.swiftbysundell.com/basics/mvvm/. Acesso em: 23 nov. 2025.

ROCHABRUN, James. Avoiding Massive View Controllers by Refactoring. *Medium*, 2020. Disponível em: https://medium.com/cocoaacademymag/avoiding-massive-view-controllers-by-refactoring-ffb6a55dfa42. Acesso em: 14 nov. 2025.

SHAHIN, M.; BABAR, M. A. On the Role of Software Architecture in DevOps Transformation: An Industrial Case Study. *arXiv preprint*, 2020.

SHOLICHIN, Fauzi; ISA, Mohd Adham Bin; HALIM, Shahliza Abd; HARUN, Muhammad Firdaus Bin. Review of iOS architectural pattern for testability, modifiability, and performance quality. *Journal of Theoretical and Applied Information Technology*, v. 97, n. 15, p. 4021–4035, 15 ago. 2019. ISSN 1992-8645. Disponível em: http://www.jatit.org/volumes/Vol97No15/3Vol97No15.pdf. Acesso em: 12 jan. 2026.

SILVA, Hayann Gonçalves. Desenvolvimento do software “Stickerz”: um aplicativo para troca de figurinhas da Copa do Mundo. 2025. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) — Escola Politécnica e de Artes, Pontifícia Universidade Católica de Goiás, Goiânia, 13 jun. 2025. Disponível em:

https://repositorio.pucgoias.edu.br/jspui/bitstream/123456789/9028/1/TCC2_Hayann_Goncalves_Silva.pdf. Acesso em: 12 jan. 2026.

SKRYPCHENKO, Mykyta. Comparison of architectural patterns within iOS applications = Порівняння архітектурних паттернів у застосунках iOS. 2024. Trabalho de conclusão de curso (Master of Software Engineering) — American University Kyiv, Kyiv, 2024. Disponível em: https://er.auk.edu.ua/handle/234907866/41. Acesso em: 12 jan. 2026.

SOMMERVILLE, Ian. *Software Engineering*. 10. ed. Harlow: Pearson Education, 2019. ISBN-13: 978-8543024974.

SOUZA, Gustavo Storck Andrade de. Desenvolvimento de um aplicativo iOS para gestão de eventos com foco em qualidade. 2020. 72 f. Monografia (Trabalho de Conclusão de Curso Bacharel em Engenharia de Computação) — Universidade Federal do Espírito Santo, Vitória, ES, Brasil, 03 jun. 2020. Disponível em: https://nemo.inf.ufes.br/wp-content/papercite-data/pdf/desenvolvimento_de_um_aplicativo_ios_para_gestao_de_eventos_com_foco_em_qualidade_2020.pdf. Acesso em: 20 nov. 2025.

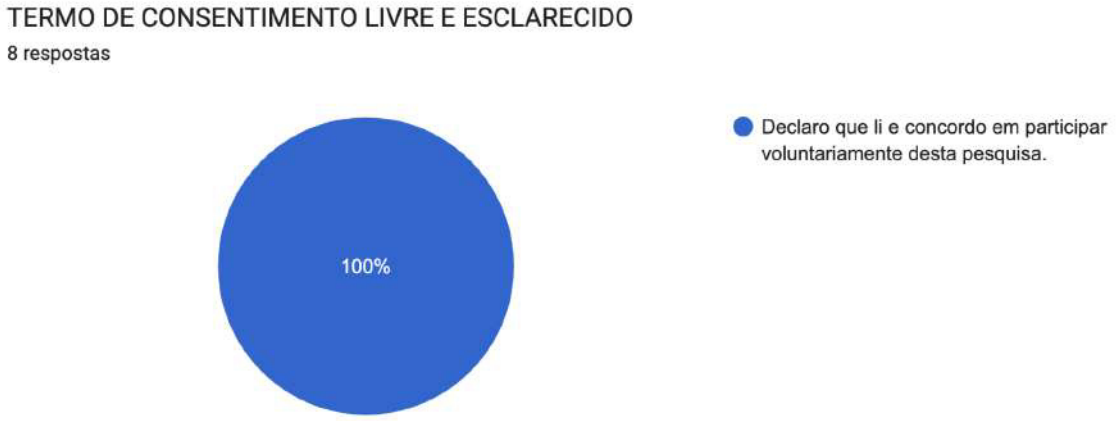
STACK OVERFLOW. Developer Survey 2023. *Stack Overflow*, 2023. Disponível em: https://survey.stackoverflow.co/2023/. Acesso em: 09 nov. 2025.

THAIYA, Mbugua Samuel; KORONGO, Julia; MBUGUA, Samuel; et al. On Software Modular Architecture: Concepts, Metrics and Trends. *International Journal of Computer & Organization Trends*, v. 12, n. 1, p. 3–10, 2022. Disponível em: https://ijcotjournal.org/archive/ijcot-v12i1p302. Acesso em: 05 jan. 2026.

APÊNDICES

APÊNDICE A – Questionário sobre arquiteturas de software no desenvolvimento iOS

Figura 10 - Termo de consentimento

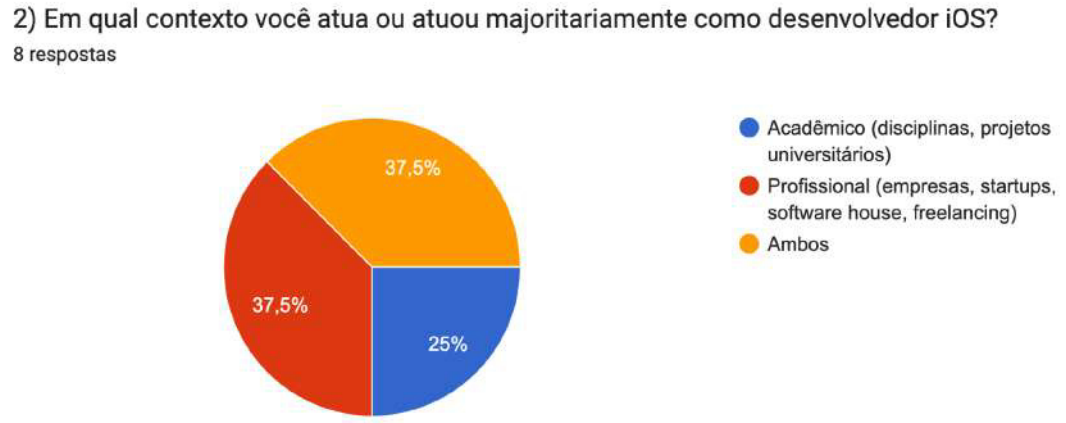


Fonte: Google Forms (2026), adaptado pelo autor.

Figura 11 - Pergunta 1

Fonte: Google Forms (2026), adaptado pelo autor.

Figura 12 - Pergunta 2



Fonte: Google Forms (2026), adaptado pelo autor.

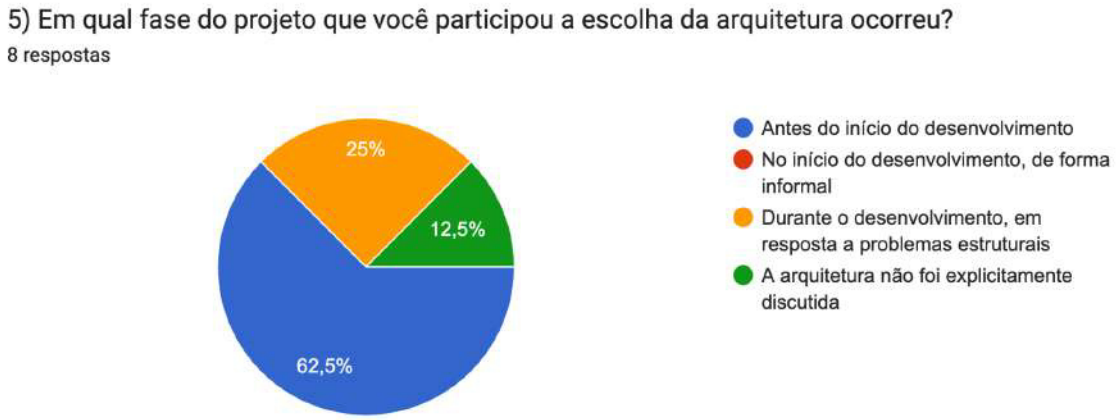
Figura 13 - Pergunta 3

Fonte: Google Forms (2026), adaptado pelo autor.

Figura 14 - Pergunta 4

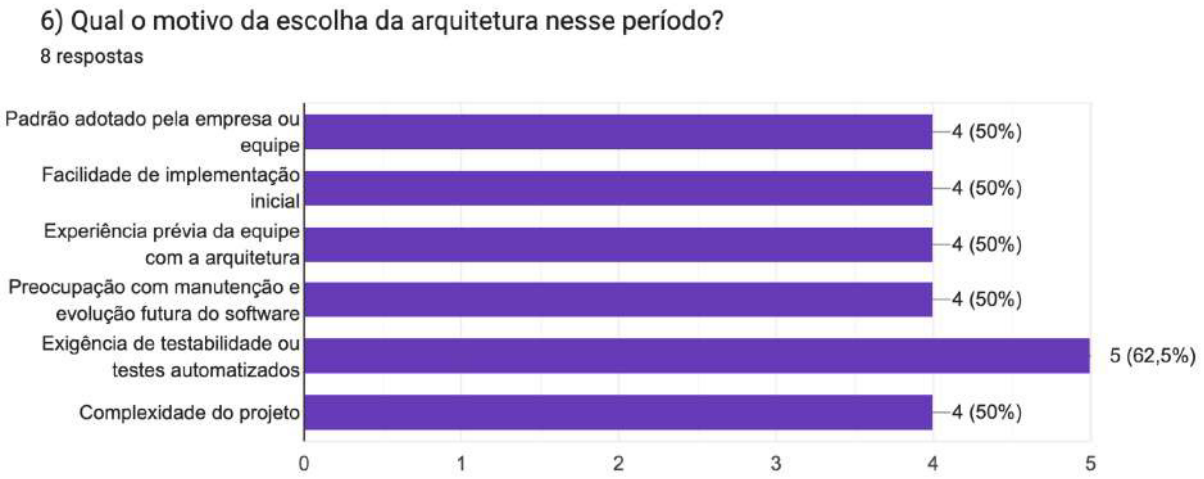
Fonte: Google Forms (2026), adaptado pelo autor.

Figura 15 - Pergunta 5



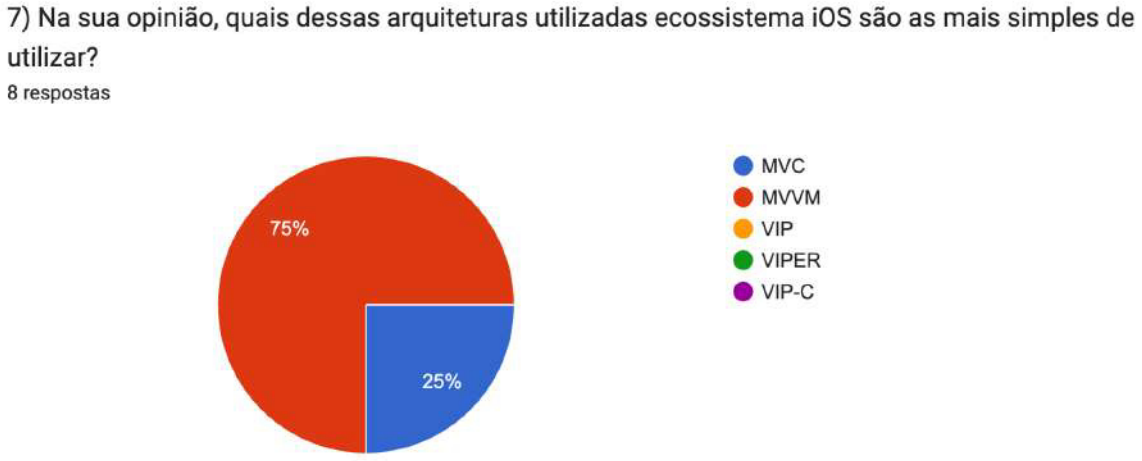
Fonte: Google Forms (2026), adaptado pelo autor.

Figura 16 - Pergunta 6



Fonte: Google Forms (2026), adaptado pelo autor.

Figura 17 - Pergunta 7



Fonte: Google Forms (2026), adaptado pelo autor.

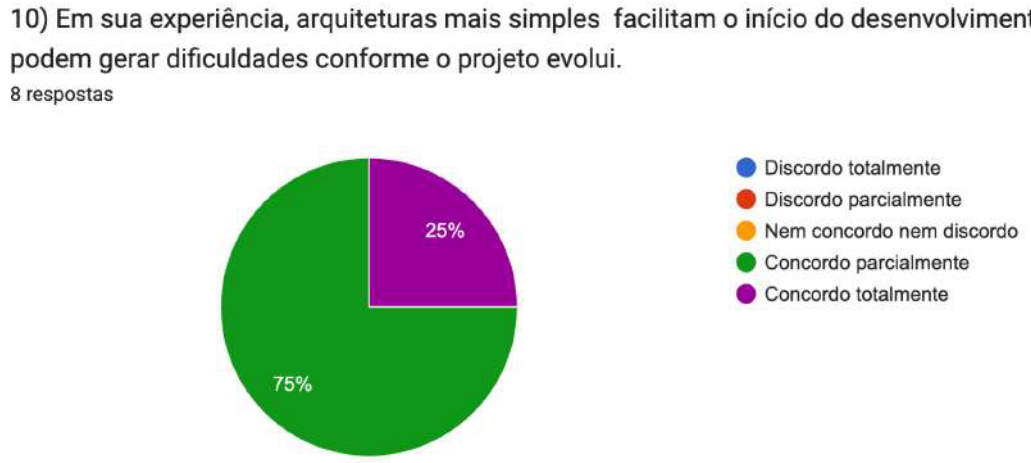
Figura 18 - Pergunta 8

Fonte: Google Forms (2026), adaptado pelo autor.

Figura 19 - Pergunta 9

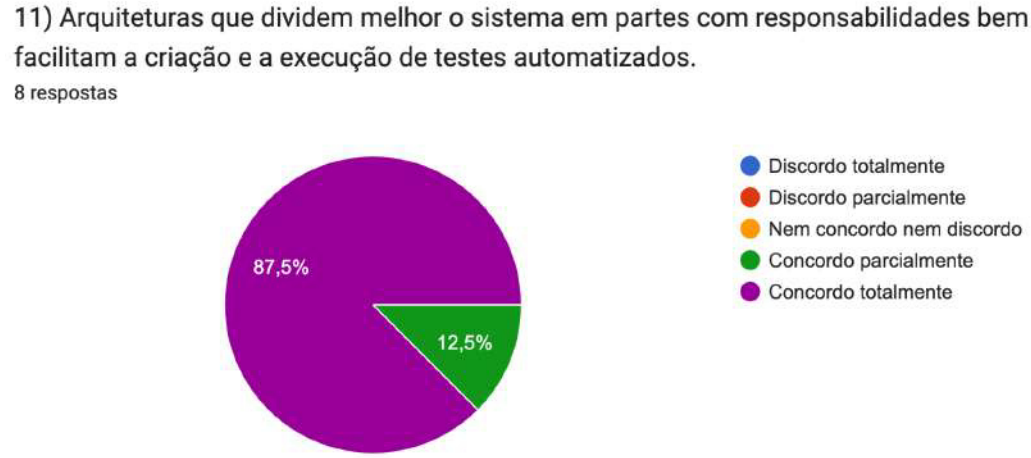
Fonte: Google Forms (2026), adaptado pelo autor.

Figura 20 - Pergunta 10



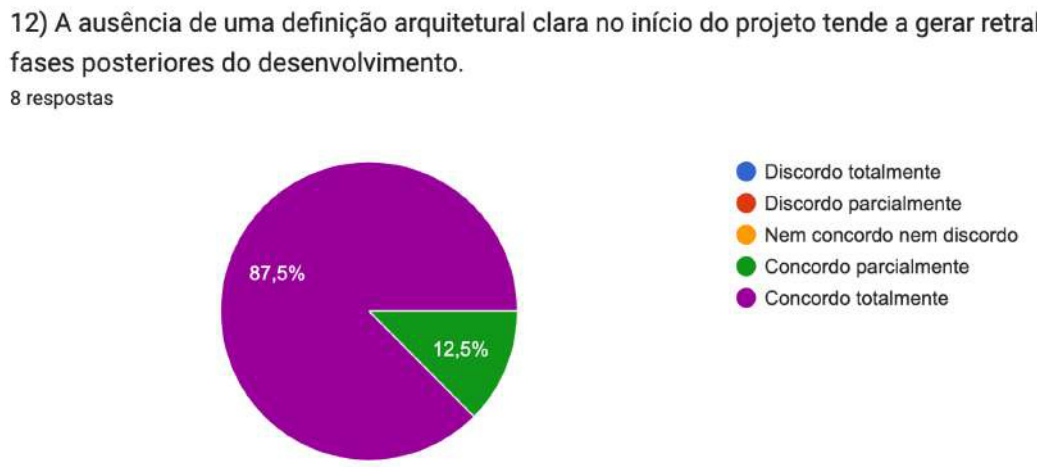
Fonte: Google Forms (2026), adaptado pelo autor.

Figura 21 - Pergunta 11



Fonte: Google Forms (2026), adaptado pelo autor.

Figura 22 - Pergunta 12



Fonte: Google Forms (2026), adaptado pelo autor.

Figura 23 - Pergunta 13

Fonte: Google Forms (2026), adaptado pelo autor.

Figura 24 - Pergunta 14

Fonte: Google Forms (2026), adaptado pelo autor.

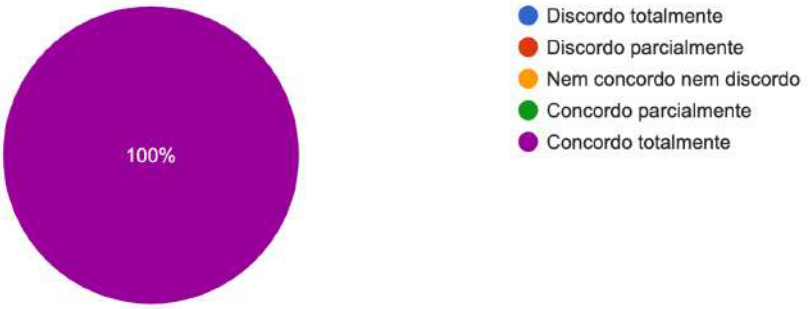
Figura 25 - Pergunta 15

Fonte: Google Forms (2026), adaptado pelo autor.

Figura 26 - Pergunta 16

16) A separação clara entre lógica de negócio, apresentação e interface gráfica facilita a escrita de testes unitários em aplicações iOS.

8 respostas

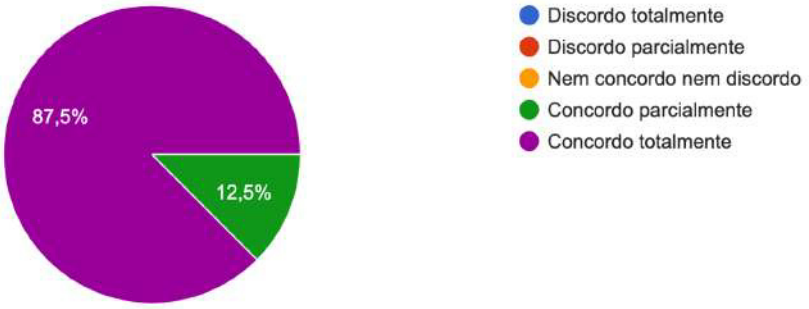


Fonte: Google Forms (2026), adaptado pelo autor.

Figura 27 - Pergunta 17

17) Decisões arquiteturais tomadas no início do projeto influenciam diretamente o esforço necessário para manter e modificar aplicações iOS ao longo do tempo.

8 respostas



Fonte: Google Forms (2026), adaptado pelo autor.