Estudo comparativo entre arquitetura MVVM e arquitetura Hexagonal para ambientes de aplicativos móveis

Anderson Tiago Izaias[1], André Marcos Silva[1]

[1]Centro Universitário Campo Limpo Paulista (UNIFACCAMP)
Jardim América – CEP 13231-230, Campo Limpo Paulista – SP – Brasil

andersontizaias@gmail.com, andre@faccamp.br

Abstract. *This paper proposes a comparative study of two development software architectures widely used by the market. The criteria base for the analysis of proposals focuses on the development of mobile applications, aiming to demonstrate their main particularities and what benefits are found in their use in software project.*

Resumo. *Este trabalho propõe o estudo comparativo de duas arquiteturas de desenvolvimento de software amplamente utilizadas pelo mercado. A base de critérios para a análise das propostas tem foco no desenvolvimento de aplicações móveis, visando demonstrar suas principais particularidades e quais benefícios são encontrados no seu uso em projetos de software.*

1. Introdução

Arquitetura de aplicação é um ramo do desenvolvimento de software preocupado com a estrutura de aplicações e mais especificamente como elas são divididas em diferentes interfaces e camadas conceituais, fluxos de controle e de dados efetuando várias operações entre componentes distintos (Eidhof, *et. al.*, 2018).

Escolher a arquitetura correta é um fator decisivo, uma vez que se torna cada vez mais desafiador construir aplicativos móveis complexos que possam ser modificados rapidamente para atender demandas de negócios, que não sofram com efeitos colaterais por conta de códigos fortemente acoplados, sejam testáveis, escaláveis e facilitem o seu desenvolvimento. Comparar e entender estas características é o caminho para encontrar o modelo mais eficaz para um bom equilíbrio entre organização, qualidade e agilidade no desenvolvimento.

2. Objetivos e Metodologia

A motivação deste trabalho é efetuar um estudo comparativo entre a arquitetura Model-View ViewModel (MVVM) e a arquitetura Hexagonal para que suas particularidades e modelos sejam comparados a fim de demostrar benefícios que a aplicação dos seus padrões pode trazer ao desenvolvimento de aplicativos móveis.

O estudo será conduzido através de pesquisa de artigos, trechos de livros e documentos que detalham características de cada uma das arquiteturas, para que os principais conceitos que as compõem sejam comparados tendo como objetivo o emprego do seu uso no desenvolvimento de aplicativos móveis.

3. Desenvolvimento

3.1. Arquitetura Model-View ViewModel (MVVM)

A arquitetura MVVM (Eidhof, *et. al*., 2018) é uma derivação da arquitetura Model View Controller (MVC) e tem o objetivo remover da camada do Controller (ViewControllers e Activities no contexto de aplicações móveis) algumas responsabilidades.

O Controller é na arquitetura MVC (em aplicações móveis) o responsável por gerenciar o ciclo de vida e estado dos elementos visuais apresentados na aplicação. Isso geralmente faz com que ele tenha uma ligação forte com os *frameworks* de elementos visuais UI (User Interface), além disso também é o responsável por interagir com o Model e consequentemente aplicar transformações de dados necessárias para refletir o resultado de interações no estado das Views. Aqui são encontrados alguns problemas desta abordagem, visto que o Controller tendo várias responsabilidades faz com que sua implementação seja complexa e seu código cresça demasiadamente, dificultando muito sua manutenção. A forte ligação com *frameworks* de elementos visuais UI traz outro problema, que é o fato de não ser possível a criação de testes unitários para cobrir as regras implementadas. A figura 1 (a), exemplifica as camadas do MVC e suas principais responsabilidades.

A principal proposta do MVVM é fazer com que a implementação da observação de alterações no Model, transformação de dados e regras de negócio sejam removidas do Controller para o ViewModel, deixando assim com única responsabilidade do Controller a gestão do ciclo de vida e dos estados dos elementos visuais.

O ViewModel passa a ter a responsabilidade de ser a ponte entre o Controller e o Model condensando em si as regras de negócio e transformações de dados necessárias para representar substancialmente o estado de apresentação da View.

Geralmente para manter a View sincronizada com o ViewModel, o MVVM propõe o uso de algum mecanismo designado para manter essa sincronia dos objetos, o Controller constrói essas ligações das propriedades da View com as propriedades do ViewModel que a representa. O mecanismo citado acima geralmente faz o uso de elementos de programação reativa podendo ser implementações nativas encontradas nas plataformas móveis ou *frameworks* criados para este fim. A figura 1 (b), mostra como estão organizadas as camadas do MVVM e suas responsabilidades.

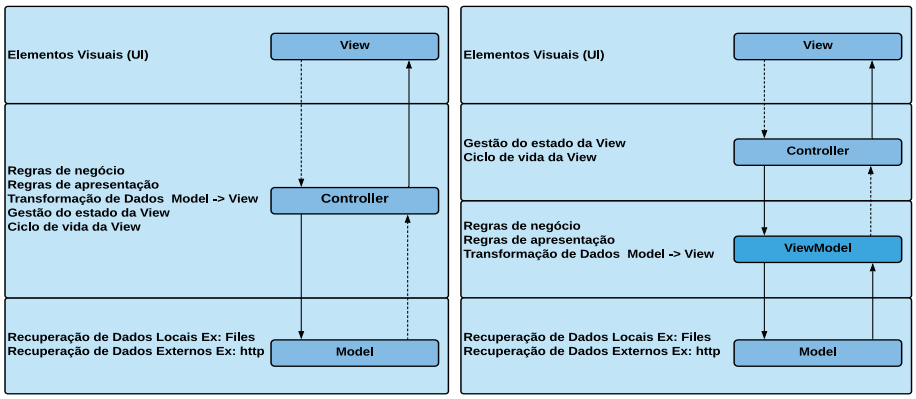

Figura 1. (a) Camadas da arquitetura MVC e suas responsabilidades e (b) Camadas da arquitetura MVVM e suas responsabilidades (Adaptado de Eidhof, *et. al*. (2018))

3.2. Arquitetura Hexagonal

A arquitetura Hexagonal (Cockburn, 2005) ou padrão de Port and Adapters foi criada para resolver problemas que geralmente são encontrados em arquiteturas que seguem o padrão de camadas. O principal objetivo dessa arquitetura é isolar de maneira efetiva as regras de negócios do sistema para que independentemente das tecnologias envolvidas na construção da interface de usuário, interações com banco de dados ou sistemas externos, geralmente é mais empregada na construção de serviços de *back-end* (APIs), mas pode ser aplicada na construção de qualquer tipo de softwares inclusive no objeto desse estudo que é o desenvolvimento de aplicativos móveis.

Como mencionado anteriormente o grande desafio na construção de softwares durante anos tem sido evitar a infiltração de regras de negócio dentro de códigos responsáveis pela criação de interfaces visuais utilizadas para a interação com os usuários, pois isso não permite a criação de testes automatizados pelo forte acoplamento dos elementos visuais com o código a ser testado.

A solução tentada repetida em muitas organizações é criar uma nova camada na arquitetura com a promessa que dessa vez realmente de verdade nenhuma regra de negócios será colocada na nova camada. Contudo, não tendo mecanismos para detectar quando esse tipo de violação ocorre, a organização encontra alguns anos depois a nova camada está desordenada e contem regras de negócios e o velho problema reaparece (Cockburn, 2005).

A arquitetura Hexagonal define conceito “dentro” e “fora” da aplicação visando evitar o emaranhamento de regras de negócios e interação com entidades externas. A regra a se obedecer é que o código pertencente a parte de dentro não deve vazar para a parte de fora da aplicação. Ela define que a aplicação deve se comunicar por portas (Ports) com agentes externos (Vernon, 2013).

Um fundamento que é apresentado pela arquitetura Hexagonal são os adaptadores (Adapters) que convertem as definições de um protocolo para um determinada tecnologia e vice-versa, ou seja podemos considerar uma interface gráfica de usuário com um exemplo de adaptador que mapeia as interações do usuário interagindo com a aplicação através de uma porta afim de alcançar as regras de negócios que fazem parte da aplicação e compõem o escopo de uma determinada funcionalidade.

O uso do desenho do hexágono para representar a arquitetura visa destacar visualmente alguns aspectos como a assimetria “dentro” e “fora” e a similar natureza das portas justamente com o intuito de afastar-se do modelo unidimensional de camadas e a presença de um número definido de diferentes portas.

A representação da arquitetura através do hexágono justamente para mostrar que o desenho permite que as pessoas que o utilizem tenho mais espaço para inserir portas e adaptadores conforme sua necessidade, não se limitando a um desenho unidimensional de camadas.

O termo Portas e Adaptadores abrange a proposta das partes do desenho onde podemos ver que uma porta é quem identifica uma conversa proposital e geralmente haverá vários adaptadores para qualquer porta, para várias tecnologias que podem ser conectadas por essa porta.

O padrão Portas e Adaptadores é escrito deliberadamente com a ideia que todas as portas são fundamentalmente iguais, que é muito benéfico e útil para pretensões de representações no nível arquitetônico, mas na implementação podem ser vistos de duas formas a qual são chamados de “primário” (*primary*) e “secundários” (*secondary*) (Loganathan, 2017). Os adaptadores também recebem nomes como “adaptadores condutores” (*driving adapters*) e “adaptadores conduzidos” (*driven adapters*) que está diretamente relacionado aos atores primários e atores secundários dos casos de uso (Spajic, 2019).

Segundo Cockburn (2005), “Um "ator primário" é um ator que conduz o aplicativo (o tira do estado quiescente para executar uma de suas funções anunciadas). Um "ator secundário" é aquele que o aplicativo direciona, seja para obter respostas ou simplesmente notificar. A distinção entre "primário" e "secundário" reside em quem aciona ou é o responsável pela conversa.”

Com base nessas observações e seguindo o diagrama de contexto de caso de uso do sistema, o desenho da arquitetura representa as portas primárias e adaptadores primários do lado esquerdo do hexágono, e as portas secundárias e adaptadores secundários ao lado direito. A figura 3, mostra a representação do desenho da arquitetura Hexagonal com suas características.

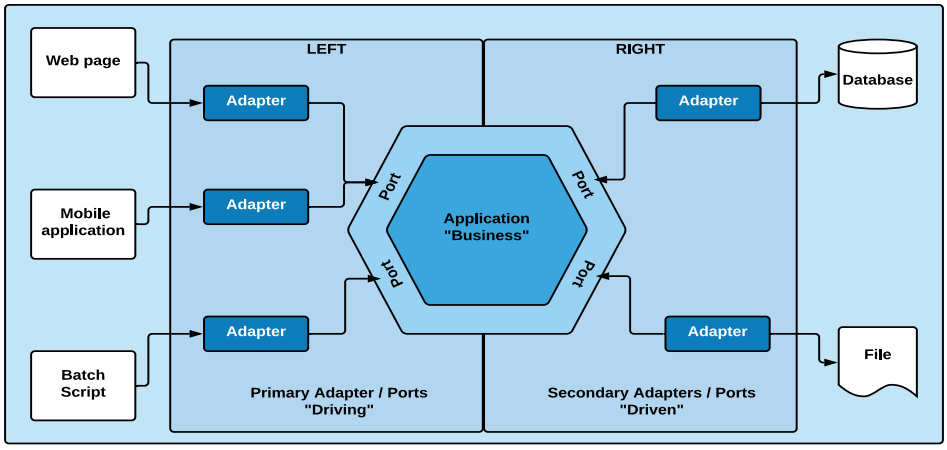

Figura 3. Arquitetura Hexagonal e suas características – Portas e Adaptadores (Adaptado de Vernon (2013))

4. Resultados

Ao confrontarmos as arquiteturas podemos notar que ambas têm representações distintas, mas compartilham de objetivos comuns, sendo principal deles a tentativa de isolar regras de negócios de particularidades de plataformas e *frameworks*. Quanto mais pura for a definição das regras de negócios e sem dependências de agentes externos mais robusto e coeso a aplicação será, pois, o esse isolamento permitirá uma cobertura efetiva das regras de negócios através de testes de unidade.

A arquitetura MVVM leva vantagem em relação a arquitetura Hexagonal pelo seu fácil entendimento (derivada do MVC) e ampla adoção no desenvolvimento de sistemas, inclusive no desenvolvimento de aplicações móveis que foco de nosso estudo.

A arquitetura Hexagonal por sua vez é mais adotada para desenvolvimento de *back-ends*, mas pode ser usada no desenvolvimento de qualquer sistema, o foco deste

trabalho justamente tem a intenção mostrar como seria sua aplicação no desenvolvimento de aplicativos móveis. Dessa forma para termos uma equivalência comparativa entre as arquiteturas que são alvo deste estudo, este trabalho propõem uma representação da arquitetura Hexagonal onde os elementos comuns ao desenvolvimento de aplicações móveis podem ser classificados. A figura 4, mostra o desenho da arquitetura Hexagonal considerando em sua organização os principais elementos encontrados no desenvolvimento de aplicações móveis com base nos elementos da arquitetura MVVM.

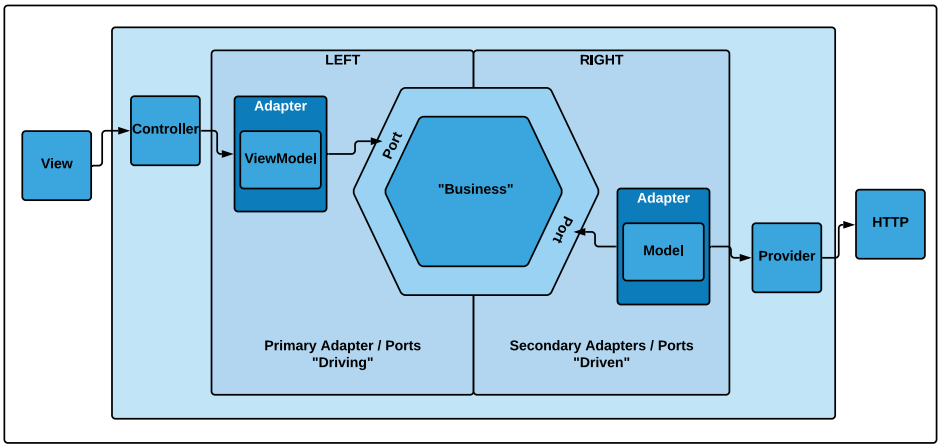

Figura 4. Arquitetura Hexagonal proposta para aplicações móveis

Como mencionado anteriormente para facilitar a comparação entre as arquiteturas o primeiro passo foi utilizar os mesmos elementos comuns ao desenvolvimento de aplicações móveis, utilizando os elementos da arquitetura MVVM pois a mesma é mais utilizada no desenvolvimento de aplicações móveis. Isso nos ajudará a entender como elementos comuns ao desenvolvimento móvel poderão se encaixar em um modelo não tão usual como a arquitetura Hexagonal.

Analisando os modelos de cada uma das arquiteturas podemos ver que a relação dos elementos e organização conta muito para a definição de responsabilidade que cada elemento terá dentro dos desenhos. Uma das principais mudanças e diferenças entre a MVVM e Hexagonal é que a definição de onde ficarão as regras de negócios. Enquanto no modelo proposto pela MVVM isso se torna responsabilidade da ViewModel na Hexagonal o modelo proposto diz que as regras de negócio devem ficar na Business e que o ViewModel passa a ter o papel de um adaptador primário que será utilizado para traduzir os eventos enviados pela Controller. O acesso a regra de negócio será feito pelo adaptador primário (ViewModel) através de uma porta primária (Interface) que tem acesso ao que chamamos de Business.

Outro detalhe na comparação entre ambas arquiteturas que é muito importante destacar é que na MVVM o Model fica sendo responsável pela recuperação de dados locais e externos, mas de forma abstrata, pois, não há qualquer definição de regras para esta camada não há definição e isso acaba ficando a cargo do desenvolvedor fazendo com que muitas vezes regras de negócios vazem para essa cada que por sua vez deveria ser responsável somente pelas interações com fontes de dados externas ou internas. Nesse quesito a arquitetura Hexagonal leva vantagem, pois o Model é tratado como um adaptador secundário e sua única função é trabalhar na tradução das interações das regras de negócios com os agentes externos ou locais.

5. Conclusão

Através deste trabalho foi possível ter uma visão mais clara sobre como as arquiteturas estudadas se relacionam com as camadas e elementos no desenvolvimento de softwares em especifico aplicações móveis. Essa comparação permitiu entender como cada uma delas trata e tenta isolar as regras de negócios da dependência de códigos especializados como *frameworks* de UI, recuperação de dados locais, recuperação de dados externos e funções nativas dos *smartphones*.

Nos resultados obtidos pela comparação ficaram evidentes que a arquitetura Hexagonal leva vantagem sobre a arquitetura MVVM, pois consegue isolar melhor as regras de negócios evitando que escapem para outros elementos da arquitetura. Mesmo não sendo amplamente utilizada no desenvolvimento de aplicações móveis pode ser empregada para orientar esse tipo de desenvolvimento e com isso trazer diversos benefícios como isolamento das regras de negócios, flexibilidade para mudanças no desenvolvimento de interfaces de usuário e testes de unidade sendo feitos sem dependências de códigos externos.

Realizar este trabalho de comparação das arquiteturas foi motivador e contribuiu para o desenvolvimento do aprendizado, principalmente no exercício do senso crítico e analítico adquiridos através dos desafios enfrentados neste projeto.

Referência Bibliográfica

Eidhof, C., *et al*. (2018). “Model-View-ViewModel+Coordinator”. App Architecture. Ed. objc.io Krugler und Eidhof GbR, 2017.

Cockburn, A. (2005). “Hexagonal Architecture – The Pattern: Ports and Adapters”. [on-line] https://alistair.cockburn.us/hexagonal-architecture. Acessado em março de 2021.

Vernon, V. (2013) “Chapter 4 Architecture: Hexagonal or Ports and Adapters”. Implementing Domain-Drive Design, United States, Ed. Pearson Education, Inc.

Spajic, Z. (2019). “Hexagonal Architecture demystified”. [on-line] https://madewithlove.com/blog/software-engineering/hexagonal-architecture-demystified. Acessado em maio de 2021.

Loganathan, P. (2017). “Hexagonal Architecture”. [on-line] https://pradeeploganathan.com/architecture/hexagonal-architecture. Acessado em maio 2021.