Arquitetura de Software MVC - Model View Controller

**Abstract.** With the evolution of technology, software has become more complex, both in terms of functionality and development and maintenance. It became necessary to use a standardization of code creation processes, to avoid recurring problems during development and execution. Design patterns and software architectures were implemented in order to mitigate such errors, and produce more efficient systems, with usability and extensibility characteristics. Given the above, this work aims to consolidate studies, research, articles and projects on the MVC (Model–View–Controller) software architecture and its various variations.

**Resumo.** Com a evolução da tecnologia, os softwares tornaram-se mais complexos, tanto em funcionalidades como em elaboração e manutenção. Tornou-se necessário o uso de padronizações de processos de criação de código, para evitar problemas recorrentes durante o desenvolvimento e execução. Os padrões de projetos e arquiteturas de software foram implementados a fim de mitigar tais erros, e produzir sistemas mais eficientes, com características de usabilidade e extensibilidade. Diante do exposto, este trabalho tem como objetivo a consolidação de estudos, pesquisas, artigos e projetos sobre a arquitetura de software MVC (Model–View–Controller) e suas diversas variações.

# 1. Introdução

Atualmente o *software* ocupa papel central na maioria das atividades humanas. É até difícil se pensar em algum tipo de atividade que não esteja de alguma forma, mesmo que indireta, dependente de *software*. Essa situação implica em grande demanda pelo desenvolvimento de *software*. E esses vêm se tornando cada vez maiores, mais sofisticados e complexos. Para que se possa lidar com esse aumento de complexidade e atender à demanda por desenvolvimento de *software* a engenharia de *software* apresenta ferramentas e métodos que visam estruturar o desenvolvimento de *software*.

A base de qualquer projeto de desenvolvimento de *software* é a arquitetura considerada para a sua construção. É nessa arquitetura que a codificação das funcionalidades do *software* efetivamente acontece. Muito conhecimento vem se acumulando sobre a arquitetura do *software*, gerando trabalhos relacionados à determinação de padrões de arquitetura que melhor se adequem à diferentes situações e necessidades.

Diversos pesquisadores vêm criando padrões de arquitetura de *software*. Trygve Reenskaug, que definiu padrões de arquitetura de projetos e identificação do modelo de arquitetura *Model*-*View*-*Controller* (MVC) e descreveu-os em uma série de relatórios e artigos como “MVC Xerox Parc 1978-79” (T. Reenskaug, 1979) e seus relatórios para a Universidade de Oslo “The original MVC reports” (T. M. H. Reenskaug, 1979 b). Após sua criação e publicação em 1978, diversas variações e aplicações foram criadas baseadas no MVC, desse modo, há muita informação conflitante disseminada atualmente, dificultando o entendimento do padrão.

Para entender uma aplicação com padrões arquiteturais e padrões de projeto é necessário descrever, entender e analisar onde e como são aplicados, como diz Ricardo Terra e Marco Túlio Valente (2010, p.1), “a utilização de padrões e a adoção de boas práticas são recomendações sempre realizadas em projetos de desenvolvimento de software. Porém, antes de descrevê-los, deve-se estudar os problemas que eles se propõem a resolver”.

Diante disso, este trabalho visa analisar e apresentar os benefícios e a aplicabilidade do padrão de arquitetura de *software* *Model*-*View*-*Controller* (MVC) e suas variações por meio de uma revisão bibliográfica sistemática.

# 2. Metodologia

A metodologia de pesquisa escolhida para esse estudo foi a revisão bibliográfica sistemática (RBS). Essa metodologia é reconhecida por ser metódica, transparente e replicável, conforme argumentam (Cook et al., 1997) e (Cooper, 1998).

A RBS conta com 3 fases principais: a entrada de informações, o processamento e a saída. No processamento, representado pelo número 2 na Figura 1, temos uma sequência de passos e atividades que processam as entradas (1) e compilam o conhecimento para a fase 3, a saída.



Figura 1. Fases da Revisão bibliográfica sistemática (Levy; Ellis. 2006)

Destacando o processo representado na Figura 1, (Levy & Ellis, 2006) destacam que

[...] revisão bibliográfica sistemática é o processo de coletar, conhecer, compreender, analisar, sintetizar e avaliar um conjunto de artigos científicos com o propósito de criar um embasamento teórico-científico (estado da arte) sobre um determinado tópico ou assunto pesquisado.

Diante da ausência de modelos específicos de estrutura para RBS para gestão de desenvolvimento e tecnologias, em 2011, Edivandro C. Conforto, Daniel C. Amaral e Sérgio L. da Silva criaram o modelo RBS *Roadmap*, desenvolvido por meio de pesquisa-ação e com base nos demais roteiros de outras áreas do conhecimento, como o modelo de Levy e Ellis (2006).

O modelo RBS *Roadmap* de Edivandro C. Conforto, Daniel C. Amaral e Sérgio L. Silva (Conforto et al., 2011), é organizado em 3 fases (Entrada, Processamento e Saída) e foi utilizado como base para a criação deste trabalho.

A fase de entrada, é o início do processo de revisão bibliográfica sistemática, onde limitamos o problema e as motivações da pesquisa. A lista a seguir apresenta a configuração da fase de entrada. São apresentados pontos iniciais da pesquisa, como problema abordado, objetivo, fontes primárias de pesquisa, *strings* utilizadas para busca, métodos de pesquisa e critérios de exclusão e inclusão de artigos, livros e teses.

* Problema: com a crescente quantidade de conteúdo publicado sobre o tema e as diversas variações do modelo criadas e disseminadas, surge a necessidade de consolidar e analisar os conteúdos relacionados ao tema.
* Objetivos: identificação, análise e seleção de estudos e informações sobre o padrão de arquitetura de software MVC, com foco em suas variações, benefícios, *frameworks* e linguagens de programação mais adequadas ao padrão.
* Fontes primárias: (T. Reenskaug & Coplien, 2009) e (T. M. H. Reenskaug, 1979).
* *String* de busca: ("model view controller" OR "mvc") AND ("software development mvc" OR "software engineering mvc") AND ("mvc variations" OR "model view controller variations").
* Critérios para inclusão: C1 - disponíveis para leitura e *download*; C2 – tema principal como a aplicação, linguagem de desenvolvimento ou demonstração lógica estrutura das camadas do MVC e variações; C3 – trabalho científico dentro do período de 1979 a 2021; C4 - apresentação de fontes de pesquisa, demonstração prática ou pesquisa-ação.
* Critérios para Exclusão: CE1 - não publicado como artigo acadêmico; CE2 – sem fonte de pesquisa; CE3 – objetivo principal não é o MVC ou variações.
* Métodos e ferramentas: Google Acadêmico; IEEE Xplore; biblioteca da universidade de Oslo; Moodle USP e SBU Unicamp.

O processamento, é a segunda fase da revisão bibliográfica sistemática, onde aplica-se os critérios da fase 1 de entrada na busca.

* Busca: Nesta etapa, é realizada a busca por período e a busca cruzada, onde o tema é relacionado com escritores, anos e linguagens de programação base.
* Análise de resultados: Etapa utilizada para a realização da leitura e análise dos resultados obtidos. Nesta etapa, foram aplicados os filtros 1, 2 e 3, criados para consolidar as fases de análise dos estudos após fase de entrada com o objetivo de descartar qualquer artigo não condizente aos demais resultados das *strings*. F1- Leitura do título, resumo e palavras-chave. Filtro 2- Leitura introdução e conclusão. Filtro 3- Leitura completa.
* Documentação: Criação de documentação com artigos encontrados que atendem a maioria ou todos os critérios de inclusão e não atendem os critérios de exclusão, quantidade de filtros atendidos da etapa de processos, data de publicação de cada artigo e filtro de palavras-chave.

Como parte da etapa de processamento foi gerada uma documentação com catálogo de controle de inserção de artigos, livros e revistas a pesquisa. Nela, temos uma ficha de leitura com informações necessárias para a criação do roteiro, como identificação (ID), título, ano, autores(as), palavras-chave, filtros, critérios de inserção, data de acesso da busca e fonte do arquivo. Com a aplicação dos critérios criados na fase 1, temos a seguinte contagem de artigos: total pesquisado 61 artigos, 49 excluídos e 12 inseridos.

Na fase de saída são analisados todos os artigos inseridos como resultado da RBS.

ID 1) (T. M. H. Reenskaug, 1979): anexa uma série de documentos que descreve suas primeiras aplicações, a definição de cada camada e pensamentos na criação do MVC. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 2) (Borini, 2015): explica e demostra uma série de variações do modelo MVC e quais são os mais populares, também demonstra a aplicação do MVC em desenvolvimentos web e quais os benefícios do seu uso. Filtros: F1, F2 e F3. Critérios: C1, C2 e C4.

ID 3) (Burbeck, 1992): descreve e cataloga aplicação do MVC em Smalltalk-80, explica o funcionamento e o limite de cada camada da aplicação, também fala dos desafios de usabilidade e das melhorias de interface. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 4) (T. Reenskaug & Coplien, 2009): aborda o tema de estrutura de código e programação orientada a objetos e estruturação baseada em MVC. Que deveria unificar as perspectivas do programador e do usuário final do código, sendo um benefício tanto para a usabilidade quanto para a compreensão das camadas do programa. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 5) (Pau Valentin Corneliu et al., 2010): demonstra que o padrão *Model* *View* *Presenter* (MVP) é um dos padrões mais utilizados para extrair a lógica de negócios fora dos elementos da interface do usuário (IU), facilitando testes unitários. Também temos a diferença do padrão para o MVC bem definidas. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 6) (Dimov et al., 2005): tem como objetivo demonstrar o MVC aplicado em desenvolvimento web. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 7) (Potel, 1996): descreve um pouco sobre desenvolvimento com *Smalltalk*, desenvolvimento MVP, benefícios do uso do MVC e controle de dados e interface do usuário. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 8) (Iyer et al., 2019): demonstra como podemos usar a variação do MVC, o PAC, para manipular e executar dados e relatórios. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 9) (T. Reenskaug, 2003): esse artigo segue com atualizações de aplicações e exemplos de usabilidade do padrão criado por Reenskaug. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 10) (Zamudio Lopez et al., 2012): demonstra a lógica e a aplicação do MVA (Model-View-Adapter) através de uma reestruturação de código. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 11) (Leff & Rayfield, 2001): manual de boas práticas e usabilidade do padrão MVC em soluções *web*. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

ID 12) (McGregor, 2004): cataloga todos os tipos de arquitetura de *software*, seus benefícios e seus desafios de aplicação. Filtros: F1, F2 e F3. Critérios: C1, C2, C3 e C4.

# 3. Arquitetura Model View Controller

O padrão estudado nesse artigo, *Model View Controller* é definido como um padrão arquitetural, que oferece uma visão de alto nível da estrutura do sistema. Segundo (Gamma et al., 1998), os padrões estruturais possuem como objetivo a definição da composição de objetos e classes na formação de estruturas maiores. E os padrões estruturais podem ser aplicados a classes ou objetos. Enquanto os padrões estruturais de objetos demonstram maneiras de complementar objetos para adquirir novas funcionalidades, os padrões estruturais de classe usam herança para definir implementações ou interfaces.

## 3.1. Criação do Padrão

Trygve Mikkejel Heyerdahl Reenskaug, nascido em 21 de junho de 1930, é um cientista de computação norueguês e professor emérito na Universidade de Oslo, na Noruega. Reenskaug foi o responsável por incluir o MVC (Model-View-Controller) em projetos de Interface Gráfica do Usuário (GUI), quando em 1978 introduziu a metodologia no Smalltalk – 79, conhecida como uma das primeiras linguagens de programação orientada a objetos dinamicamente tipada onde todos os elementos são considerados objetos, enquanto visitava o Centro de Pesquisa Xerox Palo Alto (PARC). Em 1980, Jim Althoff implementou uma versão do MVC para a biblioteca de classes Smalltalk - 80. Nesse mesmo ano, o artigo “Applications Programming in Smalltalk-80: How to use Model-View-Controller”(Burbeck, 1992) foi publicado por Steve Burbeck e assim a arquitetura ganhou mais popularidade. Mas somente em 1988, após o The Journal of Object Technology (JOT) publicar em sua revista o artigo “A Cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80” de Glenn Kresner e Stephen Pope (Krasner & Pope, 1988), o MVC foi reconhecido como um padrão de arquitetura de software.

De acordo com Reenskaug, o MVC foi criado como uma solução óbvia para um problema geral de controle dos usuários sobre suas informações, já que seu principal objetivo é isolar ao máximo as informações dispostas na camada de visualização e apresentação.

## 3.2. Definição

Reenkaug publicou em 12 de maio de 1979 uma nota onde nomeou o padrão MVC como Thing-Model-View-editor, e definiu e descreveu os termos. Em português, o termo *Thing* pode ser traduzido como coisa ou objeto e foi descrito como algo que pode ser concreto, como uma casa ou um circuito, ou algo abstrato como uma nova ideia ou uma opinião, pode ser um algo inteiro, como um computador ou somente uma parte, como um circuito elétrico, onde representam uma grande quantidade de tarefas complexas com muitos detalhes interdependentes, onde os responsáveis pelo projeto devem manter o controle dos detalhes e suas dependências, compreendendo as consequências de situações reais que podem surgir. Uma grande variedade de abstrações diferentes é usada como auxílio no controle de grandes projetos.

*Model* (Modelo) foi definida como uma representação ativa de uma abstração em forma de dados de um sistema, existem diversas formas de abstrair a mesma “*Thing*” e todas são soluções reutilizáveis, onde pode-se pensar no projeto como tendo um grande modelo subdividido em vários submodelos.

Os modelos são representados no computador como uma coleção de dados e informações em conjunto com os métodos necessários para o processamento desses dados. O ideal é que todos os modelos sejam consistentes, mas na prática não é atingível por conta de burocracia rígida, afirma Reenskaug em seu relatório “The original MVC reports”(T. M. H. Reenskaug, 1979) para a Universidade de Oslo em 1979. Todo modelo deve ter uma margem de tolerância a inconsistências, contando que as inconsistências dependam de uma quantidade mínima de tempo e esforço para corrigi-la.

Para cada Modelo, é atribuída uma ou mais *Views* (Visões), cada Visão com capacidade de mostrar representações do modelo em tela, também é tarefa da Visão demostrar as associações com o modelo correto. A implementação de Visões deve ser simplificada quando baseada no método de requisição de *Form*-*Path*-*Image*, o qual traz informações de forma dinâmica de acordo com a ação enviada por uma variável dentro do cabeçalho de requisição, a qual é associada a uma ação dentro da camada do Controlador e associada a essa Visão.

O Editor é a *interface* entre o usuário e a Visão. Disponibiliza ao usuário um sistema de comandos adequado, citando um exemplo, menus dinâmicos que mudam conforme o contexto. Fornece às Visões as entradas e mensagens necessárias ao funcionamento.

Em 10 de dezembro de 1979, Rennskaug publicou sua primeira nota com o título de *Model-View-Controllers*, como é chamado até hoje, onde fez algumas mudanças no padrão. Descreveu Modelos (Model) como conhecimento, onde pode assumir a forma de um único objeto ou representar uma estrutura de vários objetos. Os nós do modelo devem estar no mesmo nível de problema, é considerado inconsistência ter problemas de nós orientados misturados com problemas de implementação dos modelos.

Para as *Views* (Visão), descreveu como a representação visual do Modelo, destacando atributos e suprimindo outros. Resgatando os dados necessários para a apresentação do modelo, pode atualizar o modelo mandando mensagens quando atende atributos.

E o *Controllers* (Controle) como o link entre o usuário e o sistema. Auxiliando o usuário a colocar novas entradas e controlando o arranjo de *views* para cada entrada no lugar correto da tela. O controlador recebe as inserções de dados, como apresentado na Figura 2, traduz os dados para mensagens e passa as mensagens para as *views* correspondentes. Um controlador é conectado a todas as *views*, algumas delas podem ter *views* em especial, ou seja, um Editor. Um editor permite o usuário modificar a informação apresentada pela *view*. Também pode funcionar como o caminho entre o controlador e suas visões e como uma extensão do controlador. Uma vez que a edição de dados é concluída, o editor é removido e descartado.

De acordo com as conclusões de Reenskaug (1980, p.2)

[...] MVC foi concebido como uma solução geral para o problema de usuários que controlam um grande e complexo conjunto de dados. A parte mais difícil foi encontrar bons nomes para os diferentes componentes arquitetônicos. Model-View-Editor foi o primeiro conjunto. Após longas discussões, principalmente com Adele Goldberg, encerramos com os termos Model-View-Controller.

Diagrama

Descrição gerada automaticamente  
Figura 2. Representação lógica Model-View-Controller

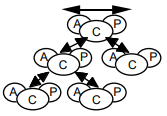
Frank Buschmann publicou em 1996 o livro “Pattern-Oriented Software Architecture” (Bushmann et al., 1996), definindo as interações entre as camadas como modelo sendo responsável por gerenciar os dados do sistema, a visão renderizando o modelo em um formato específico e o controlador que responde a entrada do usuário e realiza as interações nos objetos de modelo de dados, recebendo a entrada, validando os dados e passando para o modelo.

## 3.3. Variantes do Model View Controller

O padrão MVC, com a evolução de requisitos e sistemas cada vez mais complexos, evoluiu dando origem a variantes como Modelo Visão Controlador Hierárquico (Hierarchical model-view-controller - HMVC), Modelo Visão Apresentador (Model-view-presenter - MVP), Modelo Visão Adaptador (Model-view-adapter - MVA), Modelo Visão Modelo-visão (Model-view-viewmodel - MVVM) e o Apresentação-abstração-controle (Presentation-abstraction-control – PAC).

3.3.1- Presentation-abstraction-control

*Presentation-abstraction-control* (PAC) criado em 1987, se assemelha ao MVC também separando o sistema interativo em tríades responsáveis por funcionalidades especificas. Na PAC (Coutaz, 1997), como apresentado na Figura 3, para cada componente temos, camada de abstração que recupera e processa todos os dados inseridos e armazenados, a camada de apresentação que formata e prepara toda a apresentação de mídia, incluindo dados visuais e áudios, a camada de controle, lida com fluxos de controle e comunicação entre as camadas de componente.

  
Figura 3. Componentes do PAC são configurados baseados em outros eventos e camadas (Coutaz, 1997)

Considerando as hipóteses de (Coutaz, 1997)

[...] No estilo PAC, os componentes são agentes de três facetas que interagem por meio de conectores baseados em eventos. As facetas (apresentação, abstração, controle) são usadas para expressar perspectivas computacionais diferentes, mas complementares e fortemente acopladas da mesma funcionalidade.

Dependências de qualquer tipo são transmitidos por meio de controles. Os controles servem como mecanismo de cola para coordenação expressa, bem como transformações de modelo de dados entre o abstrato e perspectivas concretas. Além disso, o fluxo de informações entre os agentes transita por meio de controles de forma hierárquica.

O modelo, oferece como maior objetivo de benefício, a experiencia de usuário na iniciação de telas, por ter um carregamento mais rápido já que a interface do usuário, a camada de apresentação, pode ser mostrada antes da camada de abstração ser totalmente inicializada porque oferece a opção de *multithread*, separando modelo de visualização, utilizando uma estrutura hierárquica de agentes, se comunicando apenas pela camada de controle de cada tríade.

3.3.2- Hierarchical Model View Controller

O modelo *Hierarchical Model View Controller* (HMVC) é uma variação do MVC, que se assemelha muito ao PAC, publicado em 2000 no artigo “HMVC: The layered pattern for developing strong client tiers” por Jason Cai, Ranjit Kapila e Gourav Pal (Cai, 2000), neste artigo os autores afirmam que o MVC é um padrão duradouro, mas quando se trabalha com sistemas complexos, o MVC não lida com complexidades de gerenciamento de dados, de fluxos e eventos nos aplicativos. Isso acontece pela complexidade de sua estrutura, o HMVC é um conjunto de tríades de camadas de MVC que operam como um aplicativo em conjunto, como demostrado na Figura 4.

Diagrama

Descrição gerada automaticamente  
Figura 4. Representação do modelo HMVC

A adaptação promete resolver alguns problemas, criando camadas mais poderosas, focando em composição e nas *interfaces* de tríades independentes, com objetivo de desacoplamento, reutilização de código, definição da comunicação entre camadas e aumento da testabilidade.

3.2.3.3- Model View Adapter

A versão *Model View Adapter* (MVA) utilizada em projetos de grande escala de complexidade e com fluxo alto de troca de informações transforma a camada do Controlador em um *hub* de comunicação, notificando mudanças de objetos e classes e mudanças de eventos da camada de *View*, agindo como um Mediador. Seu objetivo é não sobrecarregar a camada de visualização com as constantes informações novas para o usuário. Descrevendo a Figura 5, a camada *View* recebe um evento, faz a chamada de método apropriado no *Controller*, que define conforme os parâmetros, as informações no *Model*. O *Model* tem a função de notificar quem for necessário, muitas vezes o próprio *Controller*, de sua mudança, para assim o *Controller* buscar essa informação e fazer a atualização necessária na *View* para o usuário. Para o MVA, o *Controller* pode assumir funções de formatar, traduzir e ordenar dados do Modelo.

Diagrama

Descrição gerada automaticamente  
Figura 5. Representação do modelo Model-View-adapter (Borini, 2007)

3.2.3.4- Model View Presenter

*Model View Presenter* (MVP) versão projetada com objetivos de facilitar os testes unitários automatizados, já que permite mais facilmente os testes na camada *View*, e melhorar a separação lógica das camadas da tríade com a *View*. Toda a lógica fica localizada dentro do apresentador (Presenter) que se comunica apenas com a camada *View* através da interface, como apresentado na Figura 6, então os objetos podem ser usados para testes, desde que implemente a interface de exibição.

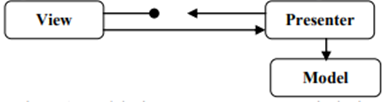
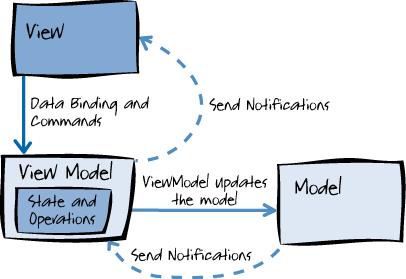


Figure 6. Modelo lógico do padrão Model View Presenter (Borini, 2007)

A camada *View*, responsável pela saída de informações para o usuário, contém e se comunica com a camada *Presenter*, que manipula o *Model* afim de recuperar e apresentar dados necessários para a lógica de negócios da camada *View*, mantendo a comunicação pela *interface*. *View* possui comunicação básica com *Model*, apenas informando qualquer mudança através de um evento de notificação (trigger), ou seja, qualquer problema de *interface* não afeta o *Model* apenas alerta as mudanças, permitindo que ele seja reutilizado em novas situações que podem ter requisitos de *interface* completamente diferentes dos previstos.

3.2.3.5- Model View ViewModel

*Model-View-ViewModel* (MVVM) neste padrão, temos três camadas principais que desempenham papéis distintos, o modelo, a visualização e o modelo de visualização, apresentadas na Figura 7. Os componentes são desacoplados, permitindo assim os componentes serem trocados, implementações que não afetam outros componentes, e que podem ser trabalhados de forma independente e testes isolados. O *ViewModel* atua como um intermediário entre a *View* e o *Model*, é responsável por toda a lógica da *View*, também recupera dados do *Model* e, em seguida, disponibiliza os dados para a exibição e pode reformatar os dados de alguma forma que os torne mais simples para o tratamento da exibição.

  
Figura 7. Representação lógica do padrão Model View ViewModel (Microsoft, 2006)

No nível mais alto, a *View* faz a ligação de dados e comandos com a camada *ViewModel*, que possui as operações e lógica de negócios, *ViewModel* "conhece" o *Model* e o atualiza, mas o *Model* não tem conhecimento do modelo de visualização (ViewModel) e o modelo de visualização não reconhece a *View*.

# 4. Conclusão

Este trabalho utilizando a metodologia de pesquisa de revisão bibliográfica sistemática foi realizado com o objetivo de identificar, analisar e selecionar estudos que abordam o uso da arquitetura de software *Model* *View* *Controller* e suas variações *Presentation Abstraction Control, Hierarchical Model View Controller, Model View Adapter, Model View Presenter, Model View ViewModel* e outras. Tendo como foco avaliar sua criação e aplicação, seus benefícios e melhores ferramentas para execução.

Os estudos inseridos foram selecionados a partir de filtros e critérios definidos na fase de Entrada e Processamento da RBS e catalogados, totalizando 12 artigos inseridos na pesquisa. Os estudos excluídos da pesquisa foram selecionados a partir de critérios de exclusão e do não oferecimento dos critérios de inclusão, 49 artigos dentre os pesquisados não foram incluídos. Para essa pesquisa foram 61 artigos encontrados através da *string* de busca no total. Importante ressaltar que os resultados da pesquisa demonstram que ainda existem algumas lacunas que os pesquisadores de engenharia e arquitetura de *software* precisam investigar para a aplicação do MVC de Reenskaug e de suas variações criadas com a popularização do padrão atualmente.

O conceito de gerenciamento em camadas do *Controller* que possibilitam o gerenciamento de requisições, validação de lógica e execução de aplicações. *Model* que armazena e executa todas as regras de negócio e os dados. E a *View*, adequando a geração dinâmica de páginas e gerenciando formulários. Esse conceito de gerenciamento foi adaptado e aplicado de maneiras diferentes em uma ampla variedade de sistemas, desde a criação do *Model-View-Controller*, temos muitas interpretações e diversas versões da aplicação que dificultam o acesso a informações oficiais da arquitetura.

Por fim, esse trabalho elenca estudos já realizados, a fim de contribuir ainda mais com a temática abordada e com o avanço do padrão *Model View Controller*.

# 5. Referências

Borini, S. (2015). Understanding Model-View-Controller. In *Gitbook*.

Borini, S. (2007) Model-View-Adapter (MVA, Mediated MVC, Model-Mediator-View). Disponível em: <https://tinyurl.com/3fhujsea>. Acesso em: 16 Jun. 2021.

Burbeck, S. (1992). Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC). *Smalltalk-80 V2*, *80*(Mvc).

Bushmann, F., Meunier, R., Rohnert, H., & Architecture, S. W. (1996). Pattern-Oriented Software Architecture. In *John Wiley&Sons* (Vol. 1).

Cai, J.; Kapila, R. e Pal, G. (2000) HMVC: The layered pattern for developing strong client tiers. InfoWorld. Disponível em: <https://tinyurl.com/3pwh2zyp>. Acesso em: 09 Jun. 2021.

Conforto, E. C., Amaral, D. C., & Silva, S. L. da. (2011). Roteiro para revisão bibliográfica sistemática: aplicação no desenvolvimento de produtos e gerenciamento de projetos. *8° Congresso Brasileiro de Gestão de Desenvolviemnto de Produto - CNGDP 2011*, *1998*.

Cook, D. J., Mulrow, C. D., & Haynes, R. B. (1997). Systematic reviews: Synthesis of best evidence for clinical decisions. In *Annals of Internal Medicine* (Vol. 126, Issue 5). <https://doi.org/10.7326/0003-4819-126-5-199703010-00006>

Cooper, H. M. (1998). Synthesizing Research: A Guide for Literature Reviews Applied Social Research Methods. In *Applied Social Research Methods Series*.

Coutaz, J. (1997). *PAC-ing the Architecture of Your User Interface*. <https://doi.org/10.1007/978-3-7091-6878-3_2>

Dimov, S., Vyas, V., Schardt, M., Faruque, A., Narla, S., & Fu, H. (2005). Customizing model-view-controller design pattern for web controlled household appliances and devices. *Proceedings of the Annual Southeast Conference*, *2*. <https://doi.org/10.1145/1167253.1167330>

Gamma, E., Vlissides, J., Johnson, R., & Helm, R. (1998). Design Patterns CD: Elements of Reusable Object-Oriented Software, (CD-ROM). *Design Patterns CD: Elements of Reusable Object-Oriented Software, (CD-ROM*.

Iyer, A., Bali, S., Kumar, I., Churi, P., & Mistry, K. (2019). Presentation Abstraction Control Architecture Pattern in Business Intelligence. *Communications in Computer and Information Science*, *1046*. <https://doi.org/10.1007/978-981-13-9942-8_62>

Krasner, G. E., & Pope, S. T. (1988). A Cookbook for Using the Model- View-Controller User Interface Paradigm in Smalltalk-80. *Joop Journal Of Object Oriented Programming*, *1*.

Leff, A., & Rayfield, J. T. (2001). Web-application development using the Model/View/Controller design pattern. *Proceedings - 5th IEEE International Enterprise Distributed Object Computing Conference*, *2001-January*(January). <https://doi.org/10.1109/EDOC.2001.950428>

Levy, Y., & Ellis, T. J. (2006). A systems approach to conduct an effective literature review in support of information systems research. *Informing Science*, *9*. <https://doi.org/10.28945/479>

McGregor, J. D. (2004). Software architecture. In *Journal of Object Technology* (Vol. 3, Issue 5). <https://doi.org/10.5381/jot.2004.3.5.c7>

MICROSOFT. ASP.NET MVC Pattern. 2006. Microsoft.com. Disponível em: <https://tinyurl.com/yckvputt>. Acesso em: 18 Jun. 2021

Pau Valentin Corneliu, Mihailescu Marius Iulian, & Stanescu Octavian. (2010). Model View Presenter Design Pattern. *Journal of Computer Science and Control Systems*, *3*(1).

Potel, M. (1996). MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. *Taligent Inc*, *C*.

Reenskaug, T. (1979). Thing-Model-View-Editor an example from a planning system. *Xerox PARC Technical Note*, *12*(May).

Reenskaug, T. (2003). The Model-View-Controller ( MVC ) Its Past and Present. *University of Oslo Draft*, *Mvc*.

Reenskaug, T., & Coplien, J. O. (2009). *The DCI Architecture : A New Vision of Object ‑ Oriented Programming*. Artima Developer.

Reenskaug, T. M. H. (1979). The original MVC reports. *Xerox PARC Technical Note*.

Zamudio Lopez, S. A., Santaolaya Salgado, R., & Fragoso Diaz, O. G. (2012). Restructuring object-oriented frameworks to model-view-adapter architecture. *IEEE Latin America Transactions*, *10*(4). <https://doi.org/10.1109/TLA.2012.6272488>