Universidade Estadual de Campinas

Faculdade de Tecnologia

Alice de Fátima da Fonseca Mantovani

Estudo Teórico da Arquitetura de Software Model View Controller

Trabalho de conclusão de curso

Limeira

2021

Alice de Fátima da Fonseca Mantovani

Estudo Teórico da Arquitetura de Software Model View Controller

Monografia apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Plínio Roberto Souza Vilela

Limeira
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Faculdade de Tecnologia
Luiz Felipe Galeffi - CRB 8/10385

M319e

Mantovani, Alice de Fatima da Fonseca, 1998-
Estudo teórico da arquitetura de *software* Model View Controller / Alice de Fatima da Fonseca Mantovani. – Limeira, SP: [s.n.], 2021.

Orientador: Plinio Roberto Souza Vilela.
Trabalho de Conclusão de Curso (graduação) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Arquitetura de software. 2. Software - Desenvolvimento. I. Vilela, Plinio Roberto Souza,1970-. II. Universidade Estadual de Campinas. Faculdade de Tecnologia. III. Título.

Informações adicionais, complementares

Título em outro idioma: Theoretical study of Model View Controller software architecture
Palavras-chave em inglês:
Software architecture
Computer software - Development
Titulação: Bacharel
Banca examinadora:
Plinio Roberto Souza Vilela [Orientador]
Gisele Busichia Baioco
Luiz Camolesi Junior
Data de entrega do trabalho definitivo: 14-12-2021

FOLHA DE APROVAÇÃO

Abaixo se apresentam os membros da comissão julgadora da sessão pública de defesa de dissertação para o Título de Bacharel em Sistemas de Informação na área de concentração, a que se submeteu a aluna Alice de Fatima da Fonseca Mantovani, em 20 de novembro de 2021 na Faculdade de Tecnologia – FT/UNICAMP, em Limeira/SP.

Prof. Dr. Plínio Roberto Souza Vilela
Presidente da Comissão Julgadora

Prof. Dra. Gisele Busichia Baioco
FT/Unicamp

Prof. Dr. Luiz Camolesi Júnior
FT/Unicamp

Ata da monografia, assinada pelos membros da Comissão Examinadora, consta no SIGA/Trabalho de Conclusão de Curso/Monografia e na Secretaria de Graduação da FT- Unicamp.

Agradecimentos

Agradeço aos meus pais e a minha irmã, por todo apoio e suporte fundamentais durante minha graduação, sem vocês eu não conseguiria.

À Universidade Estadual de Campinas e a Faculdade de Tecnologia, por todas as oportunidades durante minha jornada de formação.

Ao Prof. Dr. Plínio, pela orientação para o desenvolvimento deste trabalho de conclusão de curso.

Aos meus amigos, em especial Gustavo, Laura, Viviane e Lucca. E a República Breja Flor, que foi o meu lar nesse período de aprendizado.

Resumo

Com a evolução da tecnologia durante os anos, os *softwares* passaram a se tornar cada dia mais complexos, tanto em funcionalidades como em elaboração e manutenção. Tornou-se necessário o uso de uma padronização durante os processos de criação de código, que podem evitar problemas recorrentes durante o desenvolvimento e execução. Os padrões de projetos e arquiteturas de software começaram a ser implementados a fim de mitigar tais erros, com o intuito de produzir sistemas mais eficientes, com características de usabilidade e extensibilidade, fornecendo soluções mais completas e estruturadas ao invés de soluções imediatas.

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.

Palavras-chave: padrões de projeto, MVC, arquiteturas de software

Abstract

With the evolution of technology over the years, software has become increasingly complex, both in terms of functionalities as well as development and maintenance.
It has become necessary to use patterns throughout code creation processes, which can avoid recurring problems during development and execution. Software design patterns and architectures began to be implemented in order to mitigate such errors, in order to produce more efficient systems, with usability and extensibility characteristics, providing more complete and structured solutions instead of immediate solutions. Therefore, this work aims to consolidate studies, research, articles and projects on the MVC software architecture (Model–View–Controller) and its numerous variations.

Keywords: design patterns, MVC, software architectures

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviações

RBS	Revisão Bibliográfica Sistemática
MVC	*Model View Controller*
GUI	*Graphic User Interface*
JOT	*The Journal of Object Technology*
HMVC	Modelo Visão Controlador Hierárquico
MVP	Modelo Visão Apresentador
MVA	Modelo Visão Adaptador
MVVM	Modelo Visão Modelo-visão
POJO	*Plain Old Java Objects*
XHTML	*Extensible Markup Language*
HTML	*HyperText Markup Language*
JSF	*JavaServer Faces*
YAML	*YAML Ain't Markup Language*
CLR	*Common Language Runtime*
CLI	*Interface de Linha de Comando*
TDD	*Test Driven Development*
GPS	*Global Positioning System*
JFC	*Java Foundation Classes*

Sumário

1- Introdução

1.1- Contexto

Com a constante evolução da computação e suas tecnologias, os *softwares* vêm se tornando cada vez mais complexos, não apenas nas funcionalidades requisitadas, mas também em sua elaboração, manutenção e extensão.

Qualquer mudança de requisito, sendo ela mínima, pode causar problemas de manutenção e alteração em um projeto não estruturado. Esse é um grande desafio para os especialistas em arquitetura de *software*.

Contudo, dentro da engenharia de *software*, é apresentado um conjunto de padrões e soluções que podem ser aplicadas para solucionar tais problemas.

Com o tempo, diversos pesquisadores criaram padrões de arquitetura de *software*, mas destaca-se neste trabalho, Trygve Reenskaug, que definiu padrões de arquitetura de projetos e identificação do modelo de arquitetura *Model-View-Controller* e descreveu-os em uma série de relatórios e artigos como "*MVC Xerox Parc 1978-79*" e seus relatórios para a Universidade de Oslo em 2007 "*The original MVC reports*". 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 acesso a conteúdo do padrão.

1.2- Motivação

A motivação desse trabalho reside na crescente quantidade de conteúdo publicado sobre o tema e nas diversas variações do padrão criadas e disseminadas.

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, essa monografia 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.

1.3- Objetivo

Objetivo geral desse trabalho de conclusão de curso é identificar, analisar e selecionar estudos e informações sobre o padrão de arquitetura de software MVC, com ênfase em suas variações e benefícios.

O processo conta, também, com uma exposição de linguagens de programação e respectivos *frameworks* mais adeptos ao padrão, consolidando a pesquisa em um material único de consulta.

1.4- Estrutura

O texto foi divido em 6 capítulos, são eles:

Capítulo 1, dedicado à introdução, contexto, motivação e objetivo deste trabalho de conclusão de curso.

O Capítulo 2 explica a metodologia utilizada no processo de pesquisa e análise do desenvolvimento deste trabalho.

No Capítulo 3 é feita a apresentação com conceitos teóricos utilizados e resultado das pesquisas feitas sobre o tema e sobre o autor.

O Capítulo 4 contém a busca por ferramentas de desenvolvimento que utilizam o padrão do MVC em sua estrutura.

O Capítulo 5, contém as conclusões do trabalho.

2- Metodologia

A metodologia de pesquisa escolhida para esse estudo foi a revisão bibliográfica sistemática (RBS).

A revisão sistemática é reconhecida por ser metódica, transparente e replicável, conforme argumentam Cook et al. (1997) e Cooper (1998).

A RBS busca a análise de artigos de uma determinada área, a fim de se explorar livros, teses, artigos e revistas científicas para aprimoramento de ideias, contribuindo para o desenvolvimento de uma base sólida de conhecimento com o objetivo de compartilhar embasamentos teóricos e compreensão de assuntos e pesquisas referente ao tema.

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 e 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 (2011), é organizado em 3 fases (Entrada, Processamento e Saída) e foi utilizado como base para a criação deste trabalho.

2.1- Entrada

A fase de entrada, é o início do processo de revisão bibliográfica sistemática, onde limitamos o problema e as motivações da pesquisa.

Fase 1 – Entrada

1.1- Problema. A definição do problema é o ponto inicial da revisão.

1.2- Objetivos. Os objetivos da RBS devem estar alinhados com os objetivos do projeto de pesquisa.

1.3- Fontes primárias. As fontes primárias constituem-se de artigos, periódicos ou bases de dados que serão úteis para a definição de palavras-chave, e identificação dos principais autores e artigos relevantes.

1.4- *Strings* de busca. As *strings* de busca são um conjunto de palavras e condicionais referente ao tema de pesquisa, utilizadas para filtrar estudos. A construção de *strings* de busca segue o padrão de tentativa e erro.

1.5- Critérios de Inclusão e Exclusão. Necessário um objetivo claro da pesquisa, decisão entre quais estudos devem ser incluídos ou não na pesquisa. Neste trabalho, foram considerados estudos que atendem a maioria (3 e 4) ou todos os critérios de inclusão.

1.6- Método e ferramentas. Definição do método de busca e ferramentas envolve definir as etapas para a condução das buscas, definir os filtros de busca, como será realizada a busca nas bases de dados e como os resultados serão armazenados.

A Tabela 1 demonstra a fase de entrada. Nessa tabela, 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.

RB Sistemática	
Fase 1 - Entrada	
Problema	Com a crescente quantidade de conteúdo publicado sobre o tema e nas 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 adeptas ao padrão, consolidando a pesquisa em um material único de pesquisa.
Fontes primárias	Artigo de COPLIEN, James; Reenskaug, Trygve. O. *The DCI Architecture: A New Vision of Object-Oriented Programming.* Archive.org. Artigo de REENSKAUG, Trygve. *The original MVC reports* <https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf>.
***Strings* de busca**	("*model view controller*" OR "mvc") *AND* ("*software development mvc*" *OR* "s*oftware engineering* mvc" *OR* "s*oftware developmen*t mvc" *OR* "*software engineering* mvc") *AND* ("mvc *variations*" *OR* "*model view controller variations*") *AND* ("mvc *frameworks*" OR "mvc *frameworks java*" *OR* "mvc *frameworks php*" OR "mvc *frameworks javascript*" *OR* "mvc *frameworks for mobile applications*")
Critérios para Inclusão	C1- Disponivéis para leitura e *download*; C2- Tema principal como aplicação, linguagens de desenvolvimento ou demonstração lógica estrutural das camadas do MVC e variações; C3- os artigos devem ser trabalhos cientificos como artigos e livros ou pesquisas práticas dentro do periodo de 1979 a 2021 ; C4- apresentação de fontes de pesquisa, demonstração prática ou pesquisa-ação
Critérios de Exclusão	CE1- Artigos não publicados como trabalho acadêmico; CE2- Artigos sem fonte de pesquisa; CE3- artigos cujo o objetivo principal não seja o MVC e suas variações.
Método e ferramentas	Resultados análisados e processados afim da construção deste trabalho de conclusão de curso; Ferramentas utilizadas para pesquisa: Google Acadêmico, IEEE Xplore, Biblioteca da Universidade de Oslo, Moodle USP, SBU Unicamp

Tabela 1 – Descrição Fase de entrada da Revisão Bibliográfica Sistemática

2.2- Processamento

O processamento, é a segunda fase da revisão bibliográfica sistemática, onde aplica-se os critérios da fase 1 de entrada na busca. Para a busca, usa-se o método de busca cruzada, cruzando informações como escritores e ano, após a pesquisa, é criada a documentação para controle de artigos incluídos.

Fase 2 – Processamento

2.1- Busca. Nesta etapa, é realizada a busca por período e a busca cruzada, onde o tema é relacionado com escritores, anos, linguagens de programação e *frameworks*.
2.2- Análise de resultados. Etapa utilizada para a realização da leitura e análise dos resultados obtidos. Nesta etapa, foram aplicados os filtros 1 e 2, criados para criar fases de análise dos estudos escolhidos.
2.3- Documentação. É realizada a documentação e arquivamento dos artigos selecionados nos filtros (1 e 2), bem como os resultados das buscas e filtros de leitura (Filtro 3), e a construção inicial deste trabalho.
2.4- Ameaças a validade da revisão sistemática.

Na Tabela 2, foram apresentados processos como busca e pesquisa de dados, análise dos resultados, onde aplicamos os filtros de leitura de conteúdo, e criação da documentação de auditoria da RBS, para controle de artigos encontrados, incluídos e excluídos da pesquisa.

Fase 2	
Buscas	Busca de artigos, livros, cartas e pesquisas de conclusão de curso. Baseadas na documentação original de Trygve Reenskaug de 1979, escritas para a Universidade de Oslo.
Análise dos resultados	Nesta etapa são aplicados os filtros de leitura 1, 2 e 3, com o objetivo de selecionar os artigos para os critérios. Filtro 1- Leitura do título, resumo e palavras chave. Filtro 2- Leitura introdução e conclusão. Filtro 3- Leitura completa
Documentação	Documentação de artigos encontrados, quantidade de artigos excluídos, quantidade de artigos encontrados na busca cruzada, quantidade de inclusões, quantidade de filtros de leitura.

Tabela 2 - Fase de Processamento da Revisão Bibliográfica Sistemática

Para a execução da etapa 2.3- Documentação, foi criado um 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 (Tabela 3).

Com a aplicação dos critérios criados na fase 1, temos a seguinte contabilidade de artigos (Figura 2):

Total Excluído	24
Total Inserido	37
Total Pesquisado	61

Figura 2 - Tabela de contabilidade de Inserções e Exclusões totais

A utilização de critérios e filtros são de extrema importância para manter o foco da RBS durante a leitura, análise e catalogação do estudo.

2.3- Saída

Ao todo foram analisados 61 artigos para a construção desse trabalho, a tabela abaixo (Tabela 3) contém a documentação e o tópico 2.3.1, a análise e detalhamento dos artigos inseridos.

Documentação							
ID	Títulos	Ano	Autores (as)	Palavras-chave	Filtros	Critérios	Data da busca
1	The original MVC reports	1979	Trygve Reenskkaug	*architecture* ; mvc	F(1), F(2), F(3)	C1, C2, C3, C4	18 Jun. 2021
2	Model-View-Adapter	2019	Stefano Borini	MVC *variations* ; *web* MVC	F(1), F(2), F(3)	C1, C2, C4	16 Jun. 2021.
3	Using Zend Framework 3	2019	Oleg Krivtsov	*framework* ; mvc; *design patterns*	F(1), F(2), F(3)	C1, C2, C4	23 Jun. 2021
4	The Model-View-Presenter (MVP) Pattern	2010	Microsoft patterns & practices	MVC *variations* ; MVP; *web*	F(1), F(2), F(3)	C1, C2, C4	16 Jun. 2021
5	The MVVM Pattern	2012	Microsoft patterns & practices	*MVC variations; MVVM; interface*	F(1), F(2), F(3)	C1, C2, C4	18 Jun. 2021
6	Applications Programming in Smalltalk-80(TM) How to use Model-View-Controller (MVC)	1992	Ph.D. Steve Burbeck	*interface* ; mvc; metodo	F(1), F(2), F(3)	C1, C2, C3, C4	20 Jul. 2021
7	The layered pattern for developing strong client tiers	2000	Jason Cai, Ranjit Kapila and Gaurav Pal	*User interface; web development; mvc; controller*	F(1), F(2), F(3)	C1, C2, C3, C4	09 Jun. 2021
8	The DCI Architecture: A New Vision of Object-Oriented Programming	2009	Trygve Reenskaug e James O. Coplien	MVC; *web* ; camadas; *controller*	F(1), F(2), F(3)	C1, C2, C3, C4	10 Sep. 2021
9	PADRÕES DE PROJETO: UMA COMPILAÇÃO DOS MAIS UTILIZADOS EM PROJETOS DE SOFTWARE	2008	ALINE DE SOUSA PEREIRA	padrão de projeto; orientação a objetos; acoplamento; solução	F(1), F(2), F(3)	C1, C2, C3, C4	19 Jun. 2021
10	The ModelView Architecture	2019	Martin FITZPATRICK	*data; view, interface; web; model-view*	F(1), F(2), F(3)	C1, C2, C3, C4	23 Aug. 2021

ID	Títulos	Ano	Autores (as)	Palavras-chave	Filtros	Critérios	Data da busca
11	Ionic Framework	2020	*Ionic*	mvc; *framework; interface*	F(1), F(2), F(3)	C1, C2, C4	27 Jul. 2021
12	JavaServer Faces Technology	2021	Oracle	*framework* ; mvc	F(1), F(2), F(3)	C1, C2, C4	22 Jun. 2021
13	Laravel: The PHP Framework for Web Artisans	2011	Taylor Otwell	*framework; UX; interface*	F(1), F(2), F(3)	C1, C2, C4	26 Jun. 2021
14	Model View Presenter Design Pattern	2010	Valentin Corneliu Pau, Marius Iulian	*Model View Presenter; design pattern*	F(1), F(2), F(3)	C1, C2, C3, C4	18 Jun. 2021
15	Model-View-Presenter Framework	2012	Slawek Lasota	MVC; MVP; *framework*	F(1), F(2), F(3)	C1, C2, C4	18 Jun. 2021
16	Customizing model-view-controller design pattern for Web controlled household appliances and devices	2005	Stefan Dimov, Vikas Vyas, Mike Schardt, Abdullah Faruque, Susrutha Narla,	*Design patterns*	F(1), F(2), F(3)	C1, C2, C3, C4	5 Jul. 2021
17	O que é um framework?	2000	Dr. Jacques Philippe Sauvé	*frameworks* ; beneficios; desenvolvimento	F(1), F(2), F(3)	C1, C2, C3, C4	25 Jun. 2021
18	Uma infra-estrutura computacional para o gerenciamento de programas de ensino individualizados	2009	Alex Fernando Orlando	gerenciamento; MVC; modelos de *software*	F(1), F(2), F(3)	C1, C2, C3, C4	25 Jun. 2021
19	MVC Design Pattern in Flutter	2020	Greg Perry	*mobile; mvc; development*	F(1), F(2), F(3)	C1, C2, C3, C4	29 Jul. 2021
20	MVP: Model-View-Presenter The Taligent Programming Model for C++ and	1996	Mike Potel	*development* ; mvc; mvp	F(1), F(2), F(3)	C1, C2, C3, C4	16 Jun. 2021

ID	Títulos	Ano	Autores (as)	Palavras-chave	Filtros	Critérios	Data da busca
21	Programming ASP.NET MVC 4	2012	Jess Chadwick, Todd Snyder e Hrusikes	*pattern, mvc, architecture*	F(1), F(2), F(3)	C1, C2, C3, C4	7 Aug. 2021
22	Presentation Abstraction Control Architecture Pattern in Business Intelligence	2019	*Aksha Iyer, Sara Bali, Ishita Kumar, Prathamesh Churi e Kamal*	*MVP; Abstraction; Controller; Business Intelligence; Software Architecture*	F(1), F(2), F(3)	C1, C2, C3, C4	7 Aug. 2021
23	The Model-View-Controller (MVC) Its Past and Present	2003	Trygve Reenskaug	*mvc pattern; development; interface*	F(1), F(2), F(3)	C1, C2, C3, C4	15 Jun. 2021
24	MVC XEROX PARC 1978-79	1979	Trygve Reenskaug	*Thing-Model-View-Editor; MVC; MVC Pattern Language*	F(1), F(2), F(3)	C1, C2, C3, C4	18 Jul. 2021
25	Restructuring Object-Oriented Frameworks into Model-View-Adapter Architecture	2012	S. A. Zamudio, R. Santaolaya and O.G. Fragoso	*frameworks, software reuse, MVA pattern*	F(1), F(2), F(3)	C1, C2, C3, C4	18 Jul. 2021
26	Scaling Web Applications with HMVC	2010	Sam de Freyssinet	hmvc; mvc; *scalability*	F(1), F(2), F(3)	C1, C2, C3, C4	18 Jun. 2021
27	2021 Developer Survey	2021	*Stack Overflow Annual Develope*	popular technologies; 2021	F(1), F(2), F(3)	C1, C2, C3, C4	15 Sep. 2021
28	MVC meets Swing	1998	Todd SUNDSTED	*framework; mvc; component*	F(1), F(2), F(3)	C1, C2, C3, C4	10 Aug. 2021
29	MVC pattern in Python: Introduction and BasicModel	2017	Giacomo Debidda	*design patterns; python*	F(1), F(2), F(3)	C1, C2, C4	7 Aug. 2021
30	MVC Patterns	2008	The Griffon Team	*MVC; MVP; MVVM; Presentation Model - View - Controller*	F(1), F(2), F(3)	C1, C2, C4	11 Aug. 2021

ID	Títulos	Ano	Autores (as)	Palavras-chave	Filtros	Critérios	Data da busca
31	O impacto da utilização de design patterns nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas?	2005	Ricardo Alexsandro de Medeiros Valentim e Plácido Antônio de Souza Neto	*Desing Patterns* ; Desenvolvimento de Softwares – métricas; *Software Engineering*	F(1), F(2), F(3)	C1, C2, C3, C4	21 Jul. 2021
32	Web-application development using the Model/View/Controller design pattern	2001	Avraham Leff; James T. Rayfield	*MVC design pattern; interactive software systems; design pattern; interactive application; flexible web-application partitioning; programming model*	F(1), F(2), F(3)	C1, C2, C3, C4	17 Jun. 2021
33	Software Architecture	2004	John D. McGregor	*Desing Patterns; Software Engineering; architecture*	F(1), F(2), F(3)	C1, C2, C3, C4	18 Jun. 2021
34	ASP.NET MVC Pattern	2021	Microsoft	*framework*	F(1), F(2), F(3)	C1, C2, C4	18 Jun. 2021
35	Architecture of the Neurath Basic Model View Controller	2006	K. Yermashov, K. M. Siemsen, K. Wolke,	*software engeneering; mvc; development*	F(1), F(2), F(3)	C1, C2, C3, C4	11 Mai. 2021
36	A COMPARISON OF MODEL VIEW CONTROLLER AND MODEL VIEW PRESENTER	2013	M. Rizwan Jameel Qureshi	*Model, View, Controller, Presenter, Architectural Pattern*	F(1), F(2), F(3)	C1, C2, C3, C4	17 Jun. 2021
37	Smalltalk - 80 Bits of History, Words of Advice	1984	Glenn Krasner	Model, View, Controller; Architectural Pattern	F(1), F(2), F(3)	C1, C2, C3, C4	11 Mai. 2021

Tabela 3 - Catálogo de inserções

2.3.1- Análise dos artigos escolhidos

Neste tópico, apresento o detalhamento dos artigos processados e inseridos no trabalho de pesquisa. Os artigos são referenciados pelo ID presente no catálogo (Tabela 3).

ID	Detalhamento
1	"*The original MVC reports*" escrito em 2007 por Trygve Reenskaug, anexa uma série de documentos que descreve suas primeiras aplicações, a definição de cada camada e pensamentos na criação do MVC. Nesse artigo também é discutido o nome do modelo, na página 2, o modelo anteriormente (1979) seria chamado de "*Thing-model-view-editor*", mas ao fim do documento, na página 14, o modelo passa a se chamar "*Model-view-controller*".
2	"*Understanding Model View Controller*" escrito em 2019 por Stefano Borini, explica e demostra uma serie 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.
3	"*Using Zend Framework 3*" escrito por Oleg Krivtsov em 2019, repositório no git com aplicações, exemplos e documentação necessária para o uso do *framework* Zend.
4	"*The model view presenter MVP pattern*" divulgado em 2010 pela Microsoft, faz um detalhamento da usabilidade do padrão, sua diferença para o padrão MVC e seu objetivo.
5	"*The MVVM pattern*" divulgado em 2012 pela Microsoft, divulga quais as motivações, benefícios e usabilidade do padrão *Model-view-ViewModel*. Também demonstra as diferenças entre o modelo de seu original, o MVC.
6	*"Applications Programming in Smalltalk-80™: How to use Model-View-Contoller* (MVC)" publicado por Steve Burbeck em 1992, tem como foco descrever e catalogar a 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.

7 "*HMVC: The layered pattern for developing strong client tiers*" publicado em 2000 por Jason Cai, Ranjit Kapila e Gaurav Pal, tem como principal objetivo levantar os principais pontos de importância do MVC e sua criação das camadas *Model*, *View* e *Controller.*

8 "*The DCI Architecture: A new vision of Object-Oriented Programming*" publicado por Trygve Reenskaug e James Coplien, 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.

9 "Padrões de projeto: Uma compilação dos mais utilizados em projetos de software" publicado por Aline Pereira em 2008, tem como objetivo demonstrar a funcionalidade e os benefícios da utilização de padrões de projeto no desenvolvimento de sistemas de *software* orientado a objetos.

10 "*The ModelView Architecture*" publicado em 2019 por Martin Fitzpatrick, o artigo demonstra as diferenças e variações da modelagem MVC com a *ModelView* e como isso afeta as demais camadas e interface do usuário.

11 "*Ionic Framework*" divulgado em 2020 pela empresa Ionic, demonstra a usabilidade do framework, documentação técnica e estrutural do MVC na ferramenta e todos seus benefícios e motivações, se tornando um ótimo manual do framework para iniciantes.

12 "*JavaServer Faces Technology documentation*" documentação oficial do framework JavaServer, publicado e divulgado pela Oracle em 2017. Nessa documentação, podemos ver exemplos da aplicação do framework, benefícios e dificuldades encontradas. Assim como a estruturação de uma aplicação baseada no padrão MVC.

13 "*The PHP Framework for web artisans*" documentação oficial do *framework* Lavarel, publicada em 2011 por Taylor Otwell. Nessa documentação, temos acesso a estrutura baseada em MVC que o *framework* usa atualmente, assim como benefícios do uso do padrão.

14 "*Model View Presenter Design Pattern*" escrito por Valentin Corneliu Pau, Marius Iulian Mihailescu e Octavian Stanescu em 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.

15 "*Model View Presenter Framework*" escrito por Slawek Lasota em 2012. O artigo tem como objetivo demonstrar a lógica, estrutura, benefícios e mudanças do padrão *Model View Presenter* baseado no padrão MVC.

16 "*Customizing model-view-controller design pattern for Web controlled household appliances and devices*" escrito por Stefan Dimov, Vikas Vyas, Mike Schardt, Abdullah Faruque, Susrutha Narla e Haihua Fu em 2005, o artigo tem como objetivo demonstrar o MVC aplicado em desenvolvimento *web*.

17 "O que é um *framework*?" escrito por Jacques Philippe Sauvé em 2000. O artigo explica os benefícios e as dificuldades da aplicação de um *framework* e como é a aplicação de um *framework* com um padrão de projeto.

18 "Uma infra-estrutura computacional para o gerenciamento de programas de ensino individualizados" escrito por Alex Orlando em 2009, a dissertação explora e avalia técnicas de engenharia de *software* e computação na busca de soluções que possam reduzir custos na criação de serviços de ensino individualizados, as quais o MVC está incluso.

19 "*MVC Design Pattern in Flutter*" escrito por Greg Perry em 2020, o objetivo desse artigo é demonstrar o MVC e suas variações aplicado no *framework* Flutter.

20 "*MVP: Model View Presenter, The Taligent Programming Model for C++ and Java*" escrito por Mike Potel em 1996, o artigo descreve um pouco sobre desenvolvimento com Smalltalk, desenvolvimento MVP, benefícios do uso do MVC e controle de dados e interface do usuário.

21 "*Programming ASP.NET MVC4*" escrito por Jess Chadwick, Todd Snyder e Hrusikesh Panda em 2012, o livro tem o objetivo de orientar e ensinar desenvolvedores a utilizar o *framework* para construir *softwares* robustos e de fácil manutenção.

22 "*Presentation Abstraction Control Architecture pattern in Business Intelligence*" escrito por Aksha Iyer, Sara Bali, Ishita Kumar, Phathamesh

Churi e Kamal Mistry em 2019, esse artigo demonstra como podemos usar a variação do MVC, o PAC, para manipular e executar dados e relatórios.

23 "*The Model View Controller (MVC) Its Past and Present*" escrito por Trygve Reenskaug em 2003, esse artigo segue com atualizações de aplicações e exemplos de usabilidade do padrão criado por Reenskaug.

24 "*MVC Xerox Parc 1978-79*" escrito por Trygve Reenkaug, anexa uma série de artigos escritos por Reenskaug, como "*Thing-Model-View-Editor an Example from a planningsystem*" de 1979 e "*Administrative Control in the shipyard*" de 2003.

25 "*Restructuring Object-Oriented Frameworks to Model-View-Adapter Architecture*" escrito por S. A. Zamudio, R. Santaolaya and O.G. Fragoso em 2012, esse artigo demonstra a lógica e a aplicação do MVA através de uma reestruturação de código.

26 "*Scaling Web applications with HMVC*" escrito por Sam Freyssinet em 2010, descreve a diferença e as semelhanças entre o MVC padrão e sua variação HMVC, e uma aplicação do HMVC em um desenvolvimento *web*.

27 "*2021 Developer Survey*" divulgado pela Stack Overflow em 2021, divulga as principais ferramentas de desenvolvimento utilizadas em 2021.

28 "*MVC meets Swing*" escrito por Todd Sundsted em 1998, o artigo informa as principais características do *framework* Java Swing, também sobre a lógica do JavaServer Faces e suas classes.

29 "*MVC pattern in Python: Introduction and BasicModel*" escrito por Giacomo Debidda em 2017, o artigo aplica o MVC a uma aplicação Python, explica a lógica de cada camada e os benefícios para a interface do usuário.

30 "*MVC patterns*" divulgado em 2008 pela Griffon, empresa criadora do *framework* Griffon, o artigo demonstra algumas variações do MVC aplicadas ao *framework* desenvolvido.

31 "O impacto da utilização de design *patterns* nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas?" escrito por Ricardo A. M. Valentim e Plácido Antônio S. Neto, o artigo estuda o impacto da utilização de padrões de

desenvolvimento e estimativas de custo e entrega, trazendo em seu conteúdo os benefícios do MVC.

32 *"Web-application development using the Model-View-Controller design pattern*" escrito por Avraham Leff e James T. Rayfield em 2001, o artigo serve como um manual de boas práticas e usabilidade do padrão MVC em soluções *web.*

33 "*Software Architecture*" escrito por John D. McGregor em 2004, o artigo cataloga todos os tipos de arquitetura de software, seus benefícios e seus desafios de aplicação.

34 "*ASP.NET MVC pattern*" divulgado pela Microsoft em 2021, divulga atualizações oficiais de funcionalidades e ferramentas dentro do *framework* ASP.NET, que usa o padrão MVC.

35 "*Architecture of the neurath basic Model View Controller*" escrito por K. Yermashov, K. M. Siemsen, K. Wolke, R.A Rasenack em 2009, o artigo introduz novas variações, distintas do MVC padrão de Trygve, para aplicações especificas com *Neurath Modeling Language*.

36 "*A comparison of Model View Controller and Model View Presenter*" escrito por M. Rizwan Jameel Qureshi em 2013, o artigo apresenta em tópicos as principais diferenças entre o MVC e sua variação MVP.

37 "*Smalltalk-80 Bits of History, Words of Advice*" escrito por Glenn Krasner, o livro tem as primeiras citações sobre as aplicações do MVC na linguagem Smalltalk-80.

2.3.2- Alertas

Ameaça a validade desta revisão podem se dar pelos seguintes pontos:

- Variabilidade de aplicações relacionadas ao MVC;
- Variação de habilidades e motivação dos desenvolvedores da prática;
- Espaço amostral muito abrangente. Mesmo assim, todos os passos de uma RBS foram analisados e seguidos.
- Nos estudos selecionados, os desenvolvedores e pesquisadores usam o MVC e suas variações a fim de trazer benéficos na manutenção e extensão de aplicações, produzindo códigos de mais qualidade.

- Quebra de links ou repositórios com falta de acesso ao material utilizado.

3- Referencial Teórico

3.1- Padrões de Projeto

Padrões de desenvolvimento de *software* são soluções que podem ser aplicadas de pequenos projetos a códigos de alta complexidade. Christopher Alexander define em seu livro *Design Pattern*, considerado o ponto inicial dos padrões de projeto, que cada padrão descreve um problema que ocorre várias vezes ao nosso redor e com isso, descrevem a solução para o problema de uma maneira que você pode usar essa solução diversas vezes sem nunca ter que fazer da mesma maneira. (ALEXANDER, Christopher, 1977, p.78).

Essa colocação de Alexander também é válida considerando padrões de arquitetura de *software*. Um método de arquitetura de desenvolvimento consiste exatamente em definir uma solução dentro de um ambiente onde se tenha um problema e uma resposta. E essa solução pode ser aplicada em diversos problemas dentro do ambiente de desenvolvimento.

Os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas. Os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização. Os padrões podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Em suma, ajudam um projetista a obter um projeto “certo” mais rápido” (GAMMA *et al.*, 2000, p.18).

Cada padrão de projeto deve descrever o nome do padrão, o problema, a solução e suas consequências. O nome deve referenciar o problema que o padrão resolve, além disso, auxilia a criar um vocabulário específico que facilita a comunicação com outros arquitetos e a documentação dos sistemas. O problema descreve o contexto de aplicação de um padrão de arquitetura. A solução descreve os elementos, seus relacionamentos, responsabilidades e colaboração. As consequências são as vantagens e desvantagens de se utilizar tais padrões no desenvolvimento de um *software*.

> “Além da escolha dos algoritmos e estruturas de dados, a arquitetura envolve: decisões sobre as estruturas que formarão o sistema, controle, protocolos de comunicação, sincronização e acesso a dados, atribuição de funcionalidade

> a elementos do sistema, distribuição física dos elementos escalabilidade e desempenho e outros atributos de qualidade.” (SHAW, D. Garlan, 1996).

3.1.1- Uso de padrões de projeto

Com a complexidade e a escalabilidade das funções dos *softwares*, faz-se necessário a adoção de uma padronização durante o processo de geração de código, evitando-se assim, problemas recorrentes encontrados durante a execução do projeto. A partir deste contexto os padrões de projetos começaram a ser aplicados à problemas clássicos encontrados durante o processo de desenvolvimento de *software* (VALENTIM, 2005).

Para um projeto alcançar sucesso, Silva (2009) destaca que além do uso de padrões, é necessário o uso de uma arquitetura de processos de criação e desenvolvimento de um *software*. Essa arquitetura tem como objetivo auxiliar na organização de componentes da aplicação, alcançando benefícios como: eficiência, independência de módulos, reutilização de soluções e facilidade de manutenção.

O livro “*Design Patterns: Elements of Reusable Object-Oriented Software*” escrito por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, descreve um catálogo para os padrões, e os divide em três categorias:

1. Criação: padrões para criação de objetos, com padrões de implementação diferentes para cada política necessária. Por exemplo a criação de uma única instância de um objeto.
2. Comportamental: padrões que determinam comportamentos e relacionamentos de objetos.
3. Estrutural: definem formas de organizar classes e objetos em um sistema.

		Propósito		
		De criação	Estrutural	Comportamental
Escopo	Classe	Factory Method	Adapter (class)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Figura 3 - Catálogo de padrões de projeto (GAMMA et al., 2000, p.26)

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 (2000), 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.

Segundo Orlando (2009), o MVC proporciona uma clara separação de interesses, o que possibilita o desenvolvimento paralelo das visões e dos controladores, o aumento da produtividade da equipe de desenvolvimento e da manutenção.

3.2- *Model View Controller*

3.2.1- Criação do padrão

Nascido em 21 de junho de 1930, Trygve Mikkejel Heyerdahl Reenskaug é um cientista de computação norueguês e professor emérito na Universidade de Oslo, na Noruega. Reenskaug foi o responsável por introduzir o MVC (Model-View-Controller) em projetos de software Graphic User Interface (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” foi publicado por Steve Burbeck onde 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, o MVC foi reconhecido como um padrão de arquitetura de software.

De acordo com Reenskaug, (2007, p.1), MVC foi criado como uma solução óbvia para um problema geral de controle dos usuários sobre suas informações vista de múltiplas perspectivas.

3.2.2- 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. Para a camada *Thing*, descreveu 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 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 devem ser consistentes, mas na prática não é atingível por conta de burocracia rígida, afirma Reenskaug em seu relatório “*The original MVC reports*” para a Universidade de Oslo em 2007. 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 *Views* 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 *header* de requisição, a qual é associada a uma ação dentro da camada *Controller* associada a essa *View*.

O Editor é a interface entre o usuário e as *Views*. Disponibiliza ao usuário um sistema de comandos adequado, citando um exemplo, menus dinâmicos que mudam conforme o contexto. Fornece às *Views* 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 Visões (*Views*), 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 Controle (*Controllers*) 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 os *inputs,* como apresentado na Figura 4, 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*.

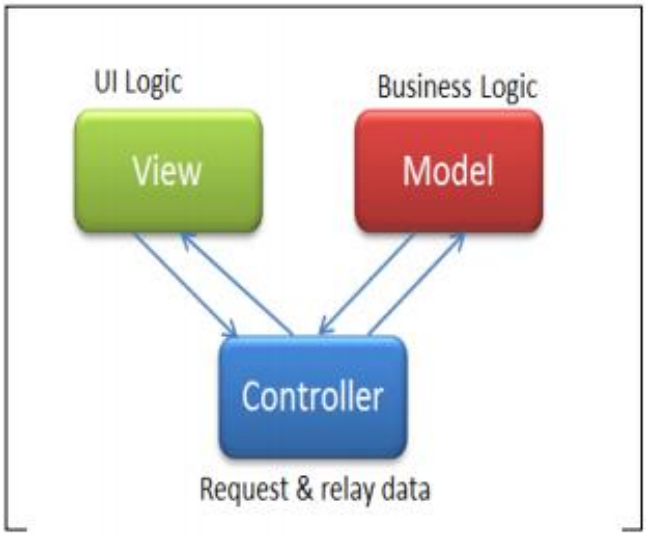

Figura 4 - Representação lógica Model-View-Controller (MICROSOFT,2021)

Frank Buschmann publicou em 1995 o livro "*Pattern-Oriented Software Architecture*", 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.

De acordo com as hipóteses de Dana Moore, Raymond Budd e Edward Benson (2007, p.67)

> [...] Desde a origem do MVC, houve muitas interpretações do padrão. O conceito foi adaptado e aplicado de maneiras muito diferentes a uma ampla variedade de sistemas e arquiteturas.

3.2.3- 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).*

3.2.3.1- *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, como apresentado na Figura 5, 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.

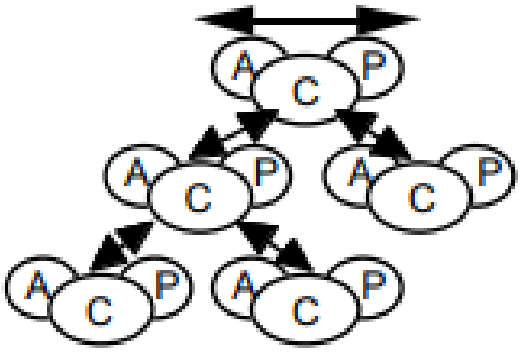

Figura 5 – Componentes do PAC são configurados baseados em outros eventos e camadas (COUTAZ, Joëlle, 1997)

Considerando as hipóteses de Coutaz (1997, p.8)

> [...] 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.

3.2.3.2- *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, 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 6.

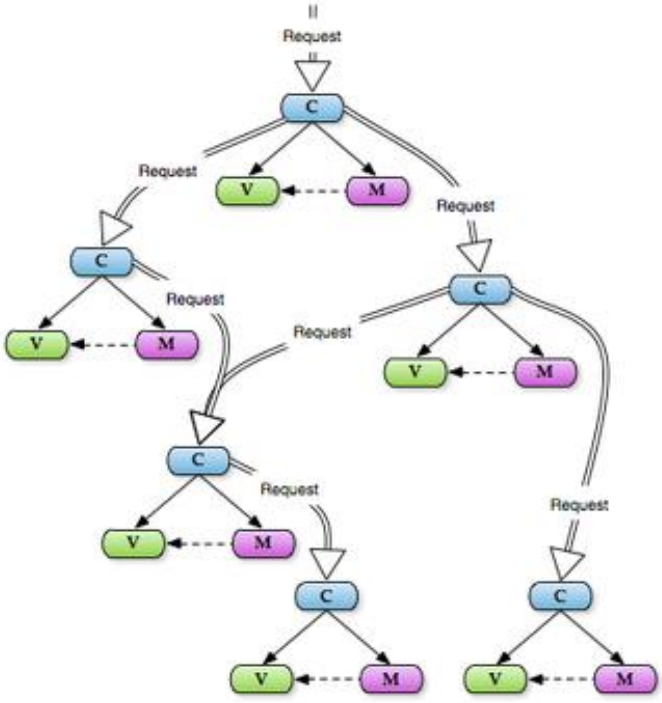

Figura 6 - Representação do modelo HMVC

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.

De acordo com Freyssinet (2010, p.1)

> [...] Cada tríade é completamente independente e pode ser executada sem a presença de nenhuma outra. Todas as solicitações feitas às tríades devem

> usar a interface do controlador, nunca carregando modelos ou bibliotecas fora de seu próprio domínio. A localização física das tríades no ambiente de hospedagem não é importante, desde que seja acessível de todas as outras partes do sistema. Os recursos distintos do HMVC encorajam a reutilização do código existente, simplifica o teste de partes distintas do sistema e garante que o aplicativo seja facilmente aprimorado ou estendido.

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 7, 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.

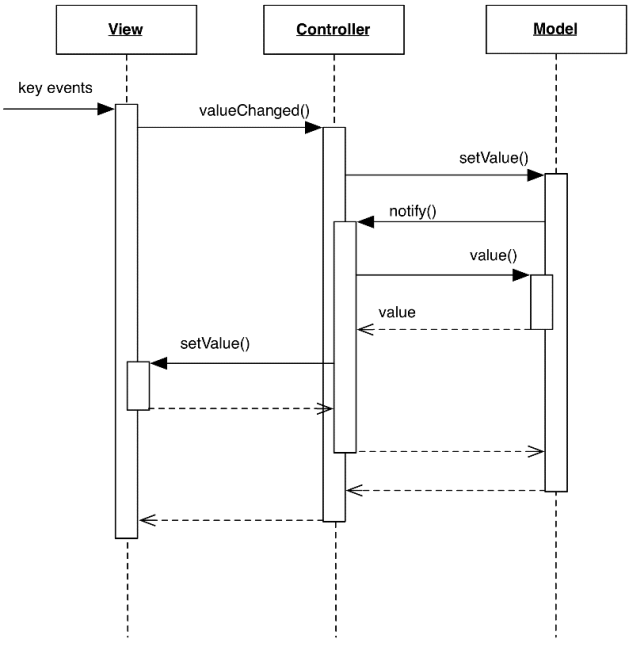

Figura 7 - Representação do modelo Model-View-adapter (Borini, 2007)

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 8, então os objetos podem ser usados para testes, desde que implemente a interface de exibição.

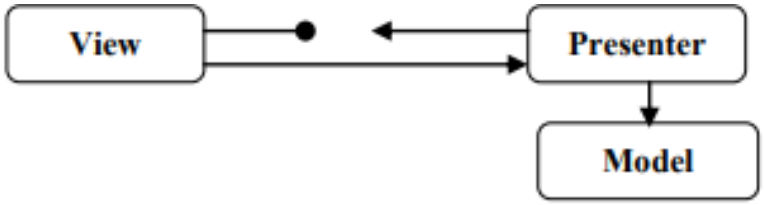

Figura 8 - Modelo lógico do padrão Model View Presenter (Borini, 2007)

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 papeis distintos, o modelo, a visualização e o modelo de visualização, apresentadas na Figura 9. 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.

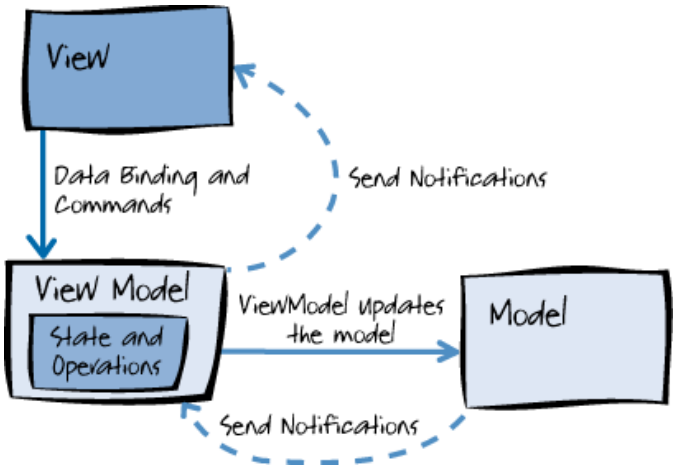

Figura 9 - Representação lógica do padrão Model View ViewModel (Microsoft, 2021)

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- Ferramentas de desenvolvimento

4.1- Linguagens

Os padrões de arquitetura de *software* podem ser descritos como uma ligação entre os componentes e funcionalidades de forma abstrata, a fim de organizar e construir códigos bem estruturados, flexíveis a aplicações variadas, seguro e de manutenção facilitada.

O padrão de arquitetura do MVC usa os métodos em camadas para melhorar a comunicação lógica entre os dados, a lógica de negócios e a interação com o

usuário. Para isso, o uso de linguagens orientadas a objetos é recomendado para o uso de arquiteturas como o MVC.

A Figura 10 apresenta a classificação de linguagens mais utilizadas nas plataformas *GitHub* e *Stack Overflow.*

Classificação | Índice	GitHub	Stack Overflow
1º	JavaScript	JavaScript
2º	Python	HTML/CSS
3º	Java	SQL
4º	TypeScript	Python
5º	C#	Java
6º	PHP	Bash/Shell/PowerShell
7º	C++	C#
8º	C	PHP
9º	Shell	TypeScript
10º	Ruby	C++

Figura 10 - Classificação de linguagens mais utilizadas no ano de 2021 (Adaptado das plataformas GitHub e Stack Overflow)

4.2- *Frameworks*

Frameworks são conjuntos e pacotes de códigos e soluções pré-definidas que podem ser utilizados no desenvolvimento dos projetos, sua proposta é facilitar a aplicação de funcionalidades, com comandos e estruturas já prontas para garantir qualidade e produtividade no projeto. Diferente de bibliotecas, os *frameworks* ditam o fluxo e controle de uma aplicação.

Nas bibliotecas, cada classe é única e independente das outras, precisando que as aplicações criem as colaborações entre as classes. Já nos *frameworks*, as

dependências e colaborações são embutidas (*wired-in interconnections*), como apresentado na Figura 11. E são diferentes dos padrões de projeto, que são mais abstratos, porque ditam com exemplos executáveis já pré-definidos em nível de código.

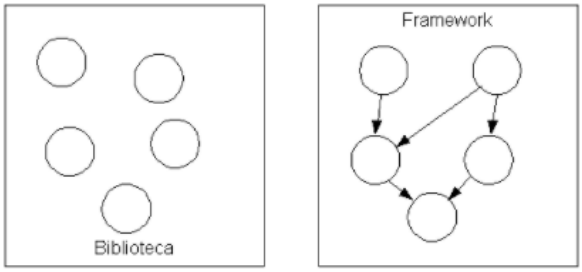

Figura 11 - Representação da diferença entre biblioteca e framework.

São designados a suportar desenvolvimento de web sites dinâmicos, aplicações e serviços *web*, os *frameworks* auxiliam os desenvolvedores com diversas linguagens. De acordo com Johnson (1997, p.3)

> [...] Aplicações que usam *frameworks* devem estar em conformidade com o modelo e design de colaboração dos *frameworks*, de modo que o próprio *framework* causa padrões nas aplicações. No entanto, os *frameworks* são mais do que apenas ideias, eles também são códigos.

4.2.1- *Frameworks Web Java*

Frameworks como *JavaServer Faces, Struts 2 e Spring MVC* são os mais utilizados para desenvolvimento *web Java* com MVC.

4.2.1.1- *JavaServer Faces*

JavaServer Faces (JSF), criado em 2004, é uma especificação Java para construção de interfaces de usuário, que inclui conjuntos de API para representar componentes

da interface e gerenciar seu estado, manipulando eventos e validações de entrada, definindo a navegação da página e oferecendo suporte à internacionalização e acessibilidade. Também inclui uma biblioteca de *tags* personalizadas, denominadas *JavaServer Pages,* que auxiliam a criação de interfaces dinamicamente baseadas em HTML e XML. Projetada para não prender o desenvolvedor à linguagens de marcação, protocolos ou dispositivo único. Com o JSF, os desenvolvedores podem escrever *views* em XHTML e através de *Data Binding*, conectá-las às classes Java. O ciclo de vida das aplicações segue uma estrutura de 6 passos, que se baseiam na arquitetura MVC de camadas na construção da aplicação, como mostrados na Figura 12.

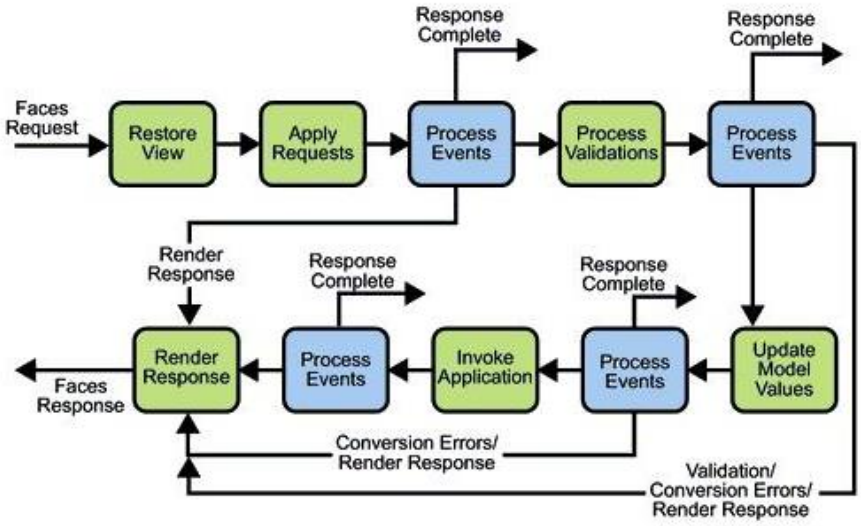

Figura 12- Ciclo de "vida" das aplicações do JavaServer Faces (Oracle, 2021)

1- Restaurar a visualização: iniciada a partir de um dado de entrada, dentro da aplicação do JSF. Essa fase é importante para a criação da visualização que liga os eventos e validadores aos componentes definidos na página. Essas informações sobre a visualização são incluídas na instância do *FacesContext*, criada ao iniciar uma requisição e destruída no fim dela. Por fim, essa instância possuirá as informações necessárias para processar a requisição.
2- Aplicação de valores de requisição: Todos os valores presentes nos componentes são extraídos e armazenados em seus tipos. Em caso de falhas na conversão, mensagens de erros são geradas e exibidas nos componentes, que será renderizado ao fim do processamento.

3- Processo de validações: Todas as validações definidas para os componentes são realizadas. Em caso de falhas na validação, uma mensagem de erro será gerada e renderizada ao fim do processamento.
4- Atualização dos valores no modelo: Com todos os componentes validados e convertidos, os dados no modelo são atualizados.
5- Invocação da aplicação: O JSF manipula eventos da aplicação para as ações serem realizadas.
6- Renderização da resposta.

4.2.1.2- *Struts 2*

O *Apache Struts 2*, lançado em 2001, é a versão atualizada que usa como base o *WebWork,* é uma estrutura de aplicativos da *web* de código aberto para o desenvolvimento *web Java* EE, que estende a API do *Java Servlet* para incentivar os desenvolvedores a adotar uma arquitetura de *Model-view-controller*, onde as camas de *Model* recebem dados e informações, a camada *View* apresenta as informações para o usuário final, e a camada *Controller* controla a interação usuário-modelo-visão. *Struts* 2 fornece uma lista de suportes e configurações, como suporte de validação, suporte *AJAX*, suporte de integração com outros *frameworks*, ações baseadas em POJO (*Plain Old Java Objects*), uma base de *plug-ins* disponibilizadas por terceiros para facilitar as aplicações, anotações utilizadas nas classes para reduzir a configuração XML e manter a configuração próxima a classe de ação e testabilidade facilitada, sendo utilizado para desenvolvimentos TDD.

4.2.1.3- *Spring MVC*

Spring MVC, desenvolvido em 2002, é uma ferramenta de aplicação com código fonte aberto bastante popular, que tem como objetivo facilitar implementações em Java EE. O *framework* consiste em um contêiner, como um gerenciador de componentes, e um conjunto de serviços de *snap-in* para interfaces de usuário, transações e persistência da *Web*. Uma parte do *Spring Framework* é o *Spring Web* MVC, um *framework* MVC extensível para criação de aplicações *Web*.
Como apresentado na Figura 13, podemos numerar alguns passos:

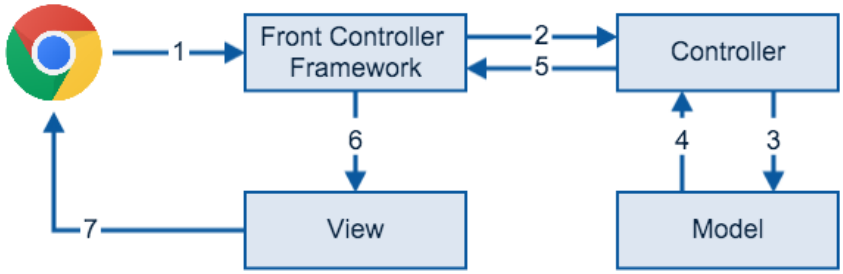

Figura 13 - Fluxo de requisição Spring MVC (Java Spring, 2021)

Através de uma URL, que envia requisições HTTP para o servidor que roda a aplicação com *Spring MVC,* quem recebe a requisição é o *controller* do *framework*. Esse controlador procura qual classe é responsável por tratar a requisição, entregando dados de entrada do browser, fazendo o papel de *controller.*

Nesta variação do MVC, o *controller* se comunica diretamente apenas com a camada do *model* e passa para a camada todos os dados, que executa todas as regras do negócio, como cálculos e validações dos dados e retorna os resultados para o *controller*. O *controller* passa as informações de saída para a camada da aplicação do *front framework*, que em conjunto com a *view* transforma a resposta em HTML e retorna ao usuário no navegador.

4.2.2- *Frameworks Web* PHP

Para desenvolvimentos web com a linguagem PHP, os *frameworks* mais populares são PHP *Zend Framework, Laravel, CakePHP.*

4.2.2.1- *Zend Framework*

Zend Framework, chegou ao mercado em 2006, tem uma coleção de mais de 60 pacotes para desenvolvimento profissional em PHP, com uma comunidade extensa de desenvolvedores, os pacotes são disponibilizados pelo *GitHub* e são abertos para o uso. Com essa variedade de pacotes e aplicações disponíveis, a *Zend Technologies* empresa distribuidora do *framework*, promete rapidez no desenvolvimento, cooperação entre o time, sites seguros com validadores e filtros de entrada e total liberdade de definição da estrutura da aplicação. Por ser um framework modular, o padrão MVC, apresentado na Figura 14 é usado da forma mais oficial, com camadas bem estruturadas e robustas.

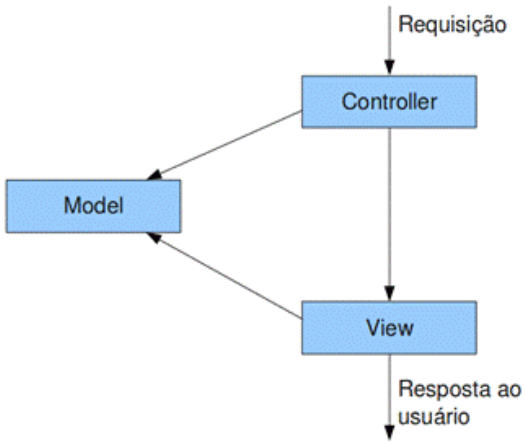

Figura 14 - Arquitetura MVC Zend Framework (Zend, 2021)

O *Zend Framework* não precisa de configurações extensas, sua linha de comando é acoplada na ferramenta *Zend_Tool*, que fornece maior produtividade na geração de arquivos da aplicação, com configuração mínima.

4.2.2.2- *Laravel*

O *Laravel*, criado em 2011, é um *framework* PHP de apoio a construção com sintaxe expressiva, disponibilizando três painéis de administração de projetos com suporte para gestão de métricas do sistema, filas e dados. Ajudando na construção de sistemas seguros, eficientes e de código limpo. Possui ferramentas como gerenciador de dependências (*Composer*), documentação intuitiva, sistema com definição concreta de rotas e sistema de *templates.*

O *Laravel* possui testes já definidos e sua estrutura possui métodos que auxiliam os testes de suas aplicações, em seu diretório, existem testes que vão de banco de dados a testes com entradas especificas.

O *framework* pode ser utilizado usando a arquitetura MVC e possui documentações específicas direcionadas apenas ao uso da arquitetura.

Na Figura 15, é representado como é o fluxo de modelo da arquitetura MVC com contexto de uma requisição HTTP, com saída em resposta HTML ou XML.

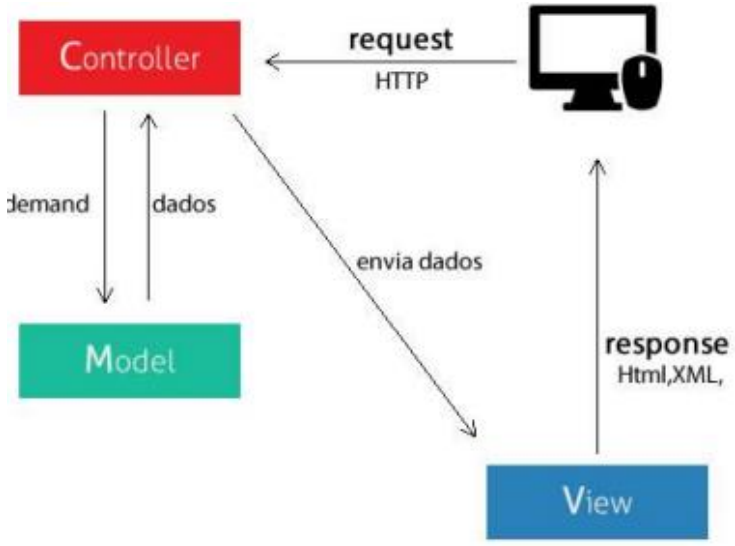

Figura 15 - Modelo de fluxo de requisição HTTP (Laravel, 2021)

Conforme o MVC descrito por Reenskaug (2007), o *controller* recebe as demandas, se comunica com o *model* para obter os dados, executa e os envia para a *view*, responsável por apresentação as informações ao usuário.

4.2.2.3- *Cake PHP*

O *framework Cake PHP,* desenvolvido em 2005, oferece uma estrutura que permite os desenvolvedores de todos os níveis a aplicarem soluções extensas e robustas rapidamente e sem perder a flexibilidade. Possui geração de código e recursos de alteração de estrutura para construção de protótipos, configuração rápida sem arquivos XML ou YAML, acesso a traduções, acesso a banco de dados, armazenamento em cache, validação e autenticação.

A Figura 16 apresenta um exemplo de requisição típica no *framework Cake PHP* começa com o usuário solicitando uma página ou recurso em sua aplicação.

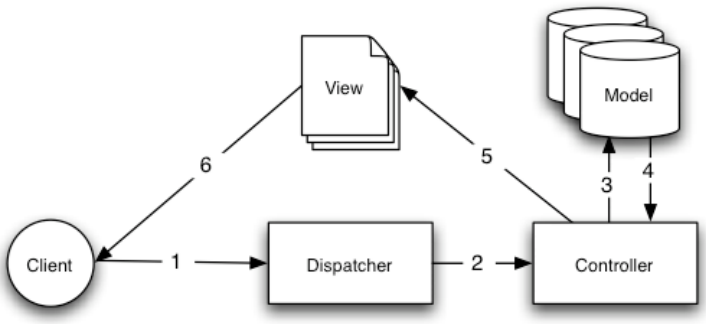

Figura 16 - Requisição básica de um client (CakePHP, 2020)

Esta solicitação é processada por um *dispatcher* (expedidor) que irá selecionar o objeto *controller* correto para lidar com a solicitação feita. Assim que a solicitação do cliente chega ao *controller* correto, este irá se comunicar como a camada *model* para processar qualquer operação de busca ou armazenamento de dados que for necessário. Após esta comunicação terminar, o *controller* continuará delegando, agora para o objeto *view* correto a tarefa de gerar uma saída resultante dos dados fornecidos pelo *model,* por fim, enviando a saída ao usuário.

4.2.3- *Framework Web C#*

Frameworks como o ASP.NET MVC são multiplataforma, podem ser usados para criação de aplicações *web, desktop* e *mobile.*

4.2.3.1- *Framework ASP.NET MVC*

O framework *ASP.NET,* teve sua primeira versão lançada em 2002*,* é uma plataforma para desenvolvimento de códigos nas linguagens C#, F# ou *Visual Basic,* de código aberto, gratuito, multiplataforma, possibilitando a construção de diversas aplicações *web, mobile* e *desktop.* O *.NET framework* tem como principais ferramentas o *Common Language Runtime* (CLR) e a biblioteca de classes. O CLR é o mecanismo de execução e controle, fornece serviços de gerenciamento de encadeamento, segurança de compartilhamento e tratamento de exceções. A biblioteca de classes fornece um conjunto de APIs e tipos de dados comuns pré-definidos, como *strings*, data e números, incluindo APIs de leitura e gravação de arquivos e conexão com banco de dados. A Figura 17 representa a arquitetura de compilação e execução do *framework,* códigos em C#, F# e *Visual basic code* compõem o início do processo de compilação*,* cada linguagem passa por um processo de execução específico mas todos recebem o tratamento das ferramentas *Common Intermediate Language* (CLI), conjunto de códigos e ações que estabelecem conexões e comandos para o compilador e *Common Language Runtime* (CLR), gerenciamento de memória e máquina virtual para executar aplicativos*.*

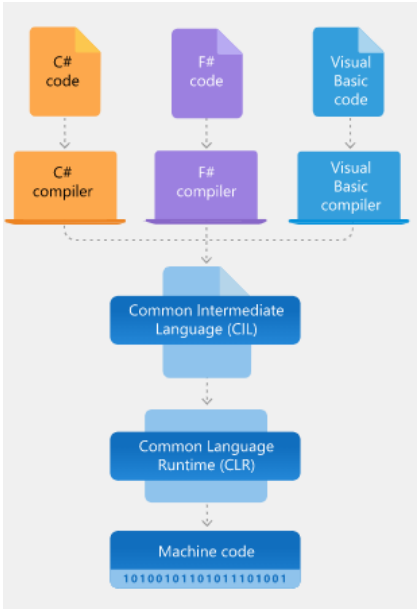

Figura 17 - Exemplo de compilação do framework ASP.NET MVC (Microsoft, 2021)

Baseado na antiga versão do *framework*, o *ASP.NET MVC framework* foi criado para a implementação do padrão *Model View Controller* para desenvolvimento *web*, que em recentes atualizações, chegou à versão do *framework .NET Core*, que unificou o *ASP.NET, ASP.NET MVC, ASP.NET Wew API* e *ASP.NET Web Pages.*

Usando a nova versão unificada, a estrutura é dividida entre as três camadas de componentes do MVC apresentadas na Figura 18, *Model, View e Controller*. Com o padrão, as solicitações de usuário são encaminhadas para um *Controller*, que é responsável por trabalhar com o *Model* para executar as ações do usuário e/ou recuperar os resultados de consultas. O *Controller* escolhe a *View* a ser exibida para o usuário e fornece-a com os dados do *Model* solicitados.

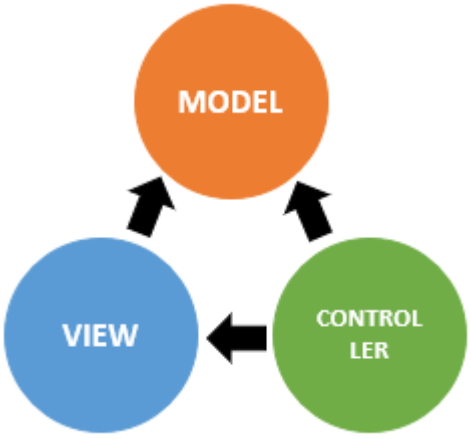

Figura 18 - Arquitetura Model View Controller.(Adaptado)

4.2.4- *Framework Web Python*

Frameworks como o Django e Web2py são populares para o desenvolvimento *web* com a linguagem Python.

4.2.4.1- Django

Django, desenvolvido em 2005, é um *framework web* na linguagem *Python*, com bibliotecas disponíveis de autenticação, administração de conteúdo e segurança, o *framework* é de código aberto e gratuito.

Usa o padrão de arquitetura de *software Model-View-Controller*, mas com algumas adequações na camada *View*. Nomeado como MTV (*Model Template View*), as camadas de *Model* e *View* mantiveram a lógica do padrão MVC, como apresentado na Figura 19, já a camada *Template* é responsável por renderizar as informações para os navegadores. E a camada *Controller* seria a própria estrutura do projeto, já que envia todas as requisições de execução para as camadas. Todas as camadas são interligadas e conversam entre si. Uma depende da outra para realizar um determinado serviço e, no final, executar a tarefa que o usuário solicitou.

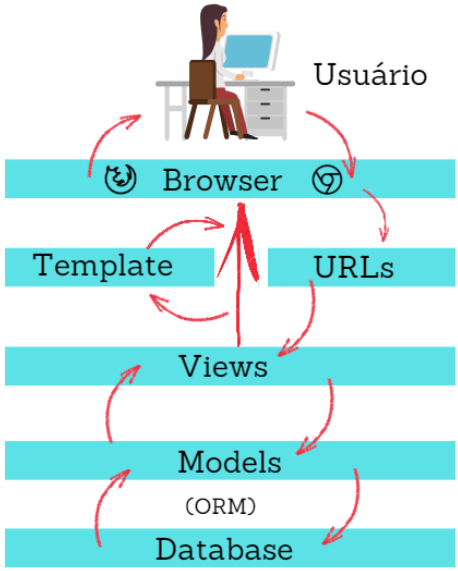

Figura 19 - Modelo lógico do modelo MTV do framework (Django, 2021)

Quando o usuário faz uma requisição pelo *browser*, utilizando uma rota, é executado um método das *Views*, que utiliza os *Models* para acessar o banco de dados e

retornar as informações. Estas informações são renderizadas pela camada de *Template* e, finalmente, renderizado para o usuário pelo navegador.

4.2.4.1- Web2py

O Web2py, teve seu lançamento em 2007, é um *framework* de código aberto e gratuito, que permite aos desenvolvedores conteúdos dinâmicos, reduzindo tempo de construção com formulários e facilitando testes unitários. O *framework* segue os padrões tradicionais do MVC (*Model View Controller*) e é autocontido, ou seja, tudo que é necessário para a construção de uma aplicação já está em suas bibliotecas internas.

O Web2py possui integrações com bancos de dados, como MSSQL, MySQL, *Postgres*, *Oracle*, *MongoDB* e *Google Big Table*. Também integra vários sub-sistemas que possibilitam a criação automática de formulários com validação automática, autenticação e autorização, gerador de códigos AJAX para melhor interação do usuário com a aplicação, upload seguro de arquivos, sistema de plugins e integração com vários padrões *Web* XML.

4.3- *Frameworks Mobile*

4.3.1- IONIC

O *framework* IONIC, criado em 2013, para desenvolvimento para aplicativos *mobile* se destaca pelo uso do padrão MVC de arquitetura e por possuir ferramentas como o CLI (Interface de Linha de Comando) que estrutura rapidamente os aplicativos do IONIC e fornece uma série de comandos úteis para os desenvolvedores. Com suporte a *React*, *Angular* e *JavaScript* construí *interfaces* e aplicativos híbridos para *iOs*, *Android*, *web* e *desktop*.

O IONIC é muito procurado por desenvolvedores mobile por utilizar tecnologia *Apache Cordova,* que possibilita criar aplicações que acessam funções nativas dos dispositivos, como GPS ou câmera.

O *model* na arquitetura do IONIC, é responsável pelo carregamento e armazenamento de dados e do envio e entrega de informações para o controlador.

O *controller* tem o objetivo de ser a ponte de comunicação entre a camada de *view* e o *model,* trazendo as requisições necessárias para atualização das *views*.

4.3.2- Flutter

Framework Flutter, desenvolvido em 2017, é um conjunto funcionalidades de desenvolvimento de interface de usuário, código aberto, na linguagem *Dart*, criado em 2017 pelo *Google*, que possibilita a criação de aplicativos compilados nativamente. As aplicações são hibridas para *Android*, *iOS*, *Windows*, *Mac*, *Linux*, *Google Fuchsia* e *Web*. O *Flutter* utiliza *widgets* como blocos de construção de funcionalidades, formando hierarquias, cada *widget* se associa ao seu *widget* pai e pode receber informações e contextos dele, facilitando e acelerando seu processamento.

Em sua arquitetura, a única constante é a separação em camadas, usando o padrão MVC. As camadas mudam a comunicação entre si, dependendo no número de aplicações e APIs chamadas, adaptando o MVC para aplicações complexas, como apresentado na Figura 20 e na Figura 21.

Um *Controller* pode ser associado à diversas *Views*, com acesso as informações do *Model* para atualizações de requisição do usuário. Mas, para reduzir o número de execuções no mesmo *Controller,* podemos conectar uma *View* a diversos controladores.

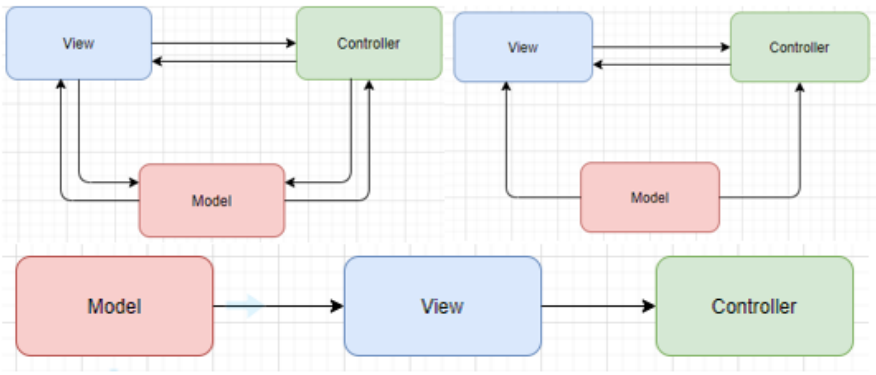

Figura 20 - Modelos lógicos da arquitetura MVC (Adaptado)

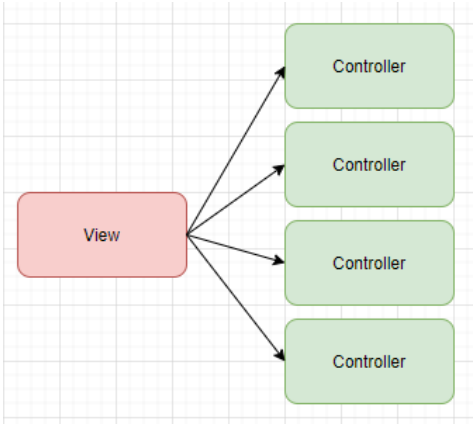

Figura 21- Modelo lógico de uma View para vários Controllers (Adaptado)

4.4- *Frameworks Desktop*

4.4.1- Ferramenta JAVA

4.4.1.1- *Swing*

Swing é um conjunto de bibliotecas e componentes disponíveis para desenvolvedores JAVA, adaptado do modelo MVC. A API *Swing* procura renderizar por conta própria todos os componentes, ao invés de delegar essa tarefa ao sistema operacional, como a maioria das outras APIs de *interface* gráfica trabalham.

Os primeiros protótipos *Swing* seguiram uma separação MVC tradicional em que cada componente tinha um componente no *Model* separado e delegava sua implementação de *interface* para objetos de *View* e *Controller* separados. Mas essa divisão não funcionava bem porque a *View* e o *Controller* exigiam um acoplamento forte e estruturado, dificultando a criação de um *Controller* genérico que não conhecesse detalhes sobre a *View*. Portanto, os dois módulos foram reunidos em único objeto de IU (interface do usuário), como apresentado na Figura 22.

Cada componente da interface do usuário, seja uma tabela, botão ou barra de rolagem possui um *Model*, um *View* e um *Controller*. Além disso, o modelo, a visualização e as peças do controlador podem mudar, mesmo enquanto o componente está em uso. O resultado é um conjunto de ferramentas de interface de usuário de flexibilidade quase incomparável.

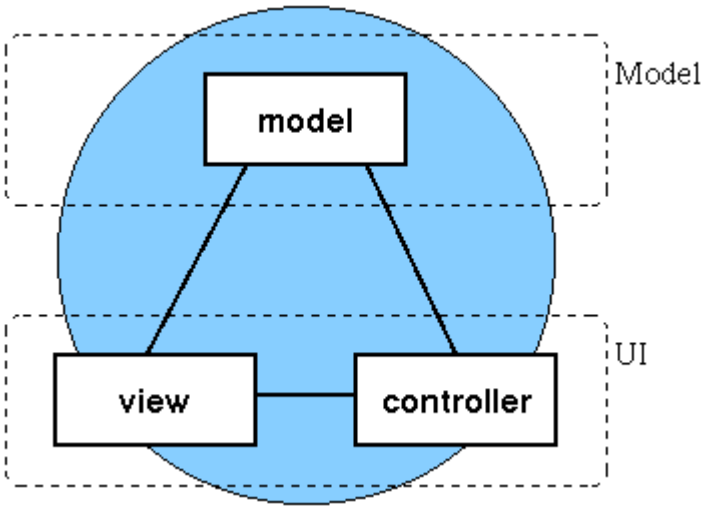

Figura 22- Componentes de interface Java Foundation Classes (JFC)

O *Swing* introduziu mecanismos que permitiam que a *interface* de cada componente fosse alterada sem mudanças no estruturais no código. Isso permite que aplicativos *Swing* emulem aparência de componentes nativos enquanto mantem benefícios da independência da plataforma, incluindo a biblioteca *JAVA Standard*, contidas no pacote *javax.swing*. Já no MVC puro de Reenskaug, sem variações, isso não é possível porque cada camada é acoplada a comunicação com o *controller.*

4.4.1.2- *Griffon*

Griffon, desenvolvido em 2008, é uma plataforma de desenvolvimento *desktop* em JAVA, compatível com ferramentas como o *Swing* e *JavaFX*. Incentiva o uso do padrão MVC e suas variações como MVVM (*Model View ViewModel*) e PMVC (*PresentationModel View Controller*).

No MVC tradicional do *Griffon*, as camadas podem se comunicar entre si, mas com algumas limitações de comunicação, por exemplo, *View* com acesso de visualização apenas ao *Model*, sem atualizar o mesmo com as informações apresentadas, limitação apresentada na Figura 23.

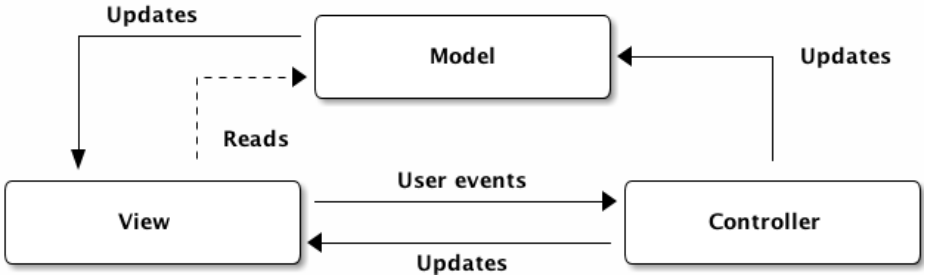

Figura 23 - Modelo lógico do padrão MVC aplicado (Griffon, 2021)

Nesta variação do MVC chamada PMVC, Figura 24, a *View* reage apenas aos dados enviados pelo *Presenter*, que vem do *Model*. A *View* pode rotear eventos do usuário para o *Presenter*, mas não tem acesso ao *Controller.* O *Controller* recebe atualizações do *Presenter* e notifica sobre qualquer alteração de estado. Apenas o *PresententationModel* conhece as duas camadas.

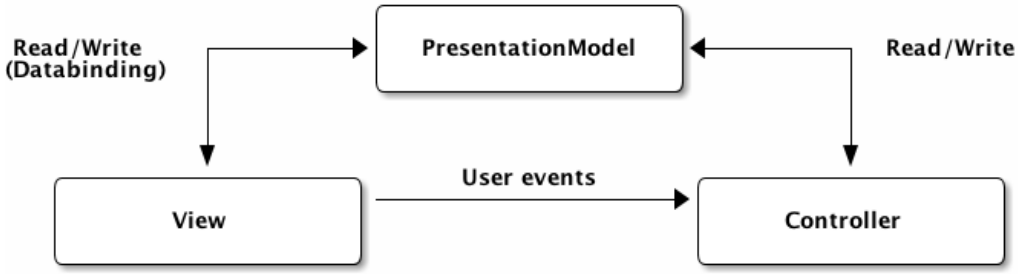

Figura 24 - Modelo lógico do padrão Presentation-Model View Controller (Griffon, 2021)

No padrão MVVM, o *ViewModel* é responsável por abstrair todas as entradas e saídas de dados da *View* enquanto executa ao mesmo tempo. Isso simplifica os testes de um aplicativo construído com esse padrão, pois o *ViewModel* tem a maior parte dos dados e do comportamento, e é separado da *View,* como apresentado na Figura 25.

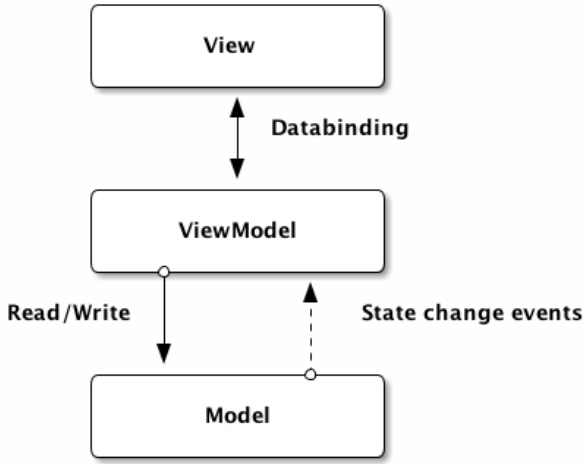

Figura 25 - Modelo lógico do padrão Model View View-Model (Griffon, 2021)

4.4.2- Ferramentas *Python*

4.4.2.1- PyQT

Qt é um *framework* multiplataforma, inicialmente criado para desenvolver aplicações *desktop*, acabou sendo portado para outras plataformas, como a *mobile*. Hoje ele suporta grandes sistemas *mobile*, como o Android e iOs.

Com a criação de *bindings* do Qt, o *framework* pode ser usado em linguagens como Java, Ruby, C# e Lua.

O PyQt, criado em 2009, é o nome do *framework* em junção a sua *binding* de *Python*, é uma ferramenta *framework* multiplataforma de desenvolvimento. Em sua versão mais atualizada PyQt 5 traz uma série de novos módulos e bibliotecas com acesso gratuito para os desenvolvedores.

A arquitetura do Qt é uma variação do modelo MVC onde a distinção entre o *View* e o *Controller* fica um pouco obscura. O Qt aceita os eventos de entrada do usuário e os delega para o *Controller* para manipulá-los. No entanto, os *controllers* também tratam da apresentação do estado atual ao usuário, colocando-os diretamente na *View*. Na arquitetura Qt, a *view* e o *controller* são mesclados criando uma arquitetura *Model ViewController*, chamada de *ModelView*.

As *ModelViews* são uma alternativa poderosa aos *widgets* de exibição padrão, que usam uma interface de modelo regular para interagir com dados, que podem ser de estruturas de dados simples a bancos de dados externos. Esse modelo isola os

dados, permitindo que sejam mantidos em qualquer estrutura necessária, enquanto a *View* atualiza a apresentação dos dados.

5- Conclusão

Este trabalho de conclusão de curso 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 Abastraction 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 Tabela 1, da fase de Entrada (2.1) e catalogados na Tabela 3, totalizando 37 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, 24 artigos dentre os pesquisados não foram incluídos. Para essa pesquisa foram 61 artigos encontrados 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 também de suas variações criadas com a popularização do padrão.

O conceito de gerenciamento em camadas de *Controller*, que possibilitam o gerenciamento de requisições, validação de lógica e execução de aplicações, *Model*, armazenando e executando regras de negócio e os dados, e *View*, adequando a geração dinâmica de páginas e gerenciando formulários, foi adaptado e aplicado de maneiras diferentes em uma ampla variedade de sistemas, como dito por Dana Moore em 2017, 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 de conclusão de curso elenca estudos já realizados, a fim de contribuir ainda mais com a temática abordada e com o avanço do padrão *Model View Controller*.

6- Referências

ALEXANDER, C. A pattern Language: town, buildings, construction. New York: Oxford University Press, 1977. 1171pp.

AKSHA IYER; BALI, Sara; ISHITA KUMAR; *et al*. Presentation Abstraction Control Architecture Pattern in Business Intelligence. Disponível em: < https://tinyurl.com/2n3be76d >. Acesso em: 2 Jul. 2021.

AMARAL, Daniel C.; Conforto, Edivandro C.; Silva, Sérgio L. Moodle USP: e-Disciplinas. edisciplinas.usp.br. Disponível em:< https://tinyurl.com/jhxybe4h>

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

BURBECK, Steve; How to use Model-View-Controller (MVC). Archive.org. Disponível em: <https://tinyurl.com/2uccaj3n>. Acesso em: 20 Jul. 2021.

CAI, RANJIT, Jason. HMVC: The layered pattern for developing strong client tiers. InfoWorld. Disponível em: <https://tinyurl.com/3pwh2zyp>. Acesso em: 09 Jun. 2021.

CHADWICK, Jess; SNYDER, Todd; PANDA, Hrusikesh. Programming ASP.NET 4:Developing real-word web Aplications with ASP.net MVC. Disponível em: < https://tinyurl.com/2p8kb8d7>. Acesso em 16 Jun. 2021

CONFORTO, E. C; AMARAL, D.C. ; SILVA, S.L. . Roteiro para Revisão Bibliográfica Sistemática: aplicação no desenvolvimento de produtos e gerenciamento de projetos. In: 8o. Congresso Brasileiro de Gestão de Desenvolvimento de Produto - CBGDP 2011, 2011, Porto Alegre-RS. 8o. Congresso Brasileiro de Gestão de Desenvolvimento de Produto - CBGDP 2011. Porto Alegre: Instituto de Gestão de Desenvolvimento de Produto, 2011.

COUTAZ, Joëlle. PAC-ing the Architecture of your User Interface; França; 1997

COPLIEN, James; Reenskaug, Trygve. O. The DCI Architecture: A New Vision of Object-Oriented Programming. Archive.org. Disponível em: <https://tinyurl.com/42zxaf37>. Acesso em: 10 Sep. 2021.

DEBIDDA, Giacomo; MVC pattern in Python: Introduction and BasicModel. Disponível em: < https://tinyurl.com/2u4npzx3>. Acesso em: 2 Sep. 2021.

DIMOV, Stefan; Faruque, Abdullah; Fu, Haihua; Narla, Susrutha; Schardt, Mike; Vyas, Vikas. *Customizing Model-View-Controller design pattern for Web Controlled Household Appliances and Devices* 2005. Disponível em: < https://dl.acm.org/doi/10.1145/1167253.1167330>. Acesso em: 09 Jun 2021.

FITZPATRICK, Martin. Using the PyQt5 ModelView Architecture to build a simple To do app. Python GUIs. Disponível em: <https://www.pythonguis.com/tutorials/modelview-architecture/>. Acesso em: 23 Aug. 2021.

GAMMA, Erich et al. Design Patterns: Elements of Reusable Object-Oriented Software. 2000 Addison-wesley, 1994. 395 p.

GITHUB. The State of the Octoverse. The State of the Octoverse. Disponível em: <https://octoverse.github.com/>. Acesso em: 7 Aug. 2021.

GRIFFON. Tutorial 5: MVC Patterns. Griffon-framework.org. Disponível em: <http://griffon-framework.org/tutorials/5_mvc_patterns.html>. Acesso em: 11 Aug. 2021.

IONIC. Cross-Platform Mobile App Development: Ionic Framework. Ionic Framework. Disponível em: <https://ionicframework.com/>. Acesso em: 27 Jul. 2021.

KRASNER, Gleen. Smalltalk-80: Bits of History, Words of Advice; California; 1983.

KRIVTSOV, OLEG. O que é o Zend Framework 3? 2019 – Using Zend Framework 3. Github.io. Disponível em: < https://tinyurl.com/t6acnn2f >. Acesso em: 23 Jun. 2021.

LARAVEL - The PHP Framework For Web Artisans. Laravel.com. Disponível em: <https://laravel.com/>. Acesso em: 26 Jun. 2021.

LASOTA, Slawek. Model-View-Presenter Framework. Mimuw.edu.pl. Disponível em: < https://tinyurl.com/25z4zpnz >. Acesso em: 18 Jun. 2021.

LEFF. A and J. T. Rayfield, "Web-application development using the Model/View/Controller design pattern," Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, 2001, pp. 118-127, doi: 10.1109/EDOC.2001.950428. Acesso em: 17 Jun. 2021.

MCGREGOR, John; Software Architecture, 2004. Disponivel em: <https://tinyurl.com/2p878eb6> Acesso em 18 Jun. 2021

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

MICROSOFT. The Model-View-Presenter (MVP) Pattern. Microsoft.com. Disponível em: < https://tinyurl.com/be45xhua >. Acesso em: 16 Jun. 2021.

MICROSOFT. ASP.NET aplicativos MVC para o .NET 6 - .NET Core.Disponível em: <https://tinyurl.com/yckvputt>. Acesso em: 18 Jun. 2021.

MICROSOFT. The MVVM Pattern. Microsoft.com. Disponível em: <https://tinyurl.com/yckvputt>. Acesso em: 18 Jun. 2021

MOORE, Dana. BUDD, Raymond. BENSON, Edward. Professional Rich Internet Applications: Ajax and Beyond; Wrok; 2007.

ORACLE, JavaServer Faces Technology. Oracle.com. Disponível em: <https://www.oracle.com/java/technologies/javaserverfaces.html>. Acesso em: 22 Jun. 2021.

ORLANDO, Alex Fernando. Uma infraestrutura computacional para o gerenciamento de programas de ensino individualizado. 2021. 181 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2009. <https://repositorio.ufscar.br/handle/ufscar/428>

PASQUARELLI, Maria Luiza Rigo. Normas para apresentação de trabalhos acadêmicos (ABNT/NBR-14724, agosto 2002) 2a edição, 2004. <https://tinyurl.com/3a76uetm>. Acesso em: 27 Nov. 2021.

PEREIRA, Aline; Padrões de Projeto: Uma compilação dos mais utilizados em projetos de Software. Faculdade de Minas -faminas-bh curso de Sistemas de informação. [s.l.: s.n.], 2008. Disponível em: < https://tinyurl.com/sunzbaub >Acesso em: 20 Jul 2021.

PERRY, Greg. MVC Design Pattern in Flutter | Flutter Community. Medium. Disponível em: < https://tinyurl.com/5r9a4j9c >. Acesso em: 29 Jul. 2021.

POTEL, Mike. MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. [s.l.]: , [s.d.]. Disponível em: <http://www.wildcrest.com/Potel/Portfolio/mvp.pdf>. Acesso em: 16 Jun. 2021.

REENSKAUG, Trygve. MVC. Universitetetioslo.no. Disponível em: <https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html>. Acesso em: 18 Jul. 2021.

REENSKAUG, Trygve. The Model-View-Controller (MVC) Its Past and Present. [s.l.]: [s.d.]. Disponível em: < https://tinyurl.com/af4kpmrv >. Acesso em: 15 Jun. 2021.

REENSKAUG, Trygve. The original MVC reports. [s.l.]: , [s.d.]. Disponível em: <https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf>. Acesso em: 18 Jun. 2021.

RIZWAN, M; QURESHI, Jameel ; SABIR, Fatima. A COMPARISON OF MODEL VIEW CONTROLLER AND MODEL VIEW PRESENTER. Sci.Int.(Lahore),2013. Disponível em: <https://tinyurl.com/wrpa42y7>. Acesso em: 25 Jun. 2021

SAM DE FREYSSINET. techPortal | Scaling Web Applications with HMVC. Archive.org. Disponível em: < https://tinyurl.com/3zekkd59 >. Acesso em: 18 Jun. 2021.

SHAW. M, D. Garlan; An Introduction to Software Architecture. Relatório TécnicoCMU-CS-94-166, Carnegie Mellon University, Pittsburgh, PA 15213-3890, Janeiro 1994.

SAUVÉ, Jacques Philippe. O que é um framework? Ufcg.edu.br. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 25 Jun. 2021.

SIEMSEN, K.H; Architecture of the Neurath Basic Model View Controller. Disponivel em: <https://tinyurl.com/2p9mm542> Acesso em: 23 Jul. 2021

SINHA S. (2017) Model, View, Controller Workflow. In: Beginning Laravel. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2538-7_10. Acesso em: 7 Aug. 2021

STACK OVERFLOW Developer Survey 2021. Stack Overflow. Disponível em: <https://insights.stackoverflow.com/survey/2021#most-popular-technologies>. Acesso em: 15 Sep. 2021.

SUNDSTED, Todd. MVC meets Swing. InfoWorld. Disponível em: <https://www.infoworld.com/article/2076632/mvc-meets-swing.html>. Acesso em: 10 Aug. 2021.

VALENTIM, Ricardo A. M.; NETO, Plácido A. S. O impacto da utilização de design patterns nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas? Revista da Farn, Natal, vol. 4, no. 1, p.63-74, dez. 2005.

ZAMUDIO, S. A. Lopez, R. Santaolaya Salgado and O. G. Fragoso Diaz, "Restructuring Object-Oriented Frameworks to Model-View-Adapter Architecture," in IEEE Latin America Transactions, vol. 10, no. 4, pp. 2010-2016, June 2012, doi: 10.1109/TLA.2012.6272488.