UNIVERSIDADE EVANGÉLICA DE GOIÁS

BACHARELADO EM ENGENHARIA DE SOFTWARE

OTÁVIO LEMES DE OLIVEIRA

**SISTEMA GERENCIADOR DE BANCO DE DADOS**

Orientado à governança empresarial

UNIVERSIDADE EVANGÉLICA DE GOIÁS

BACHARELADO EM ENGENHARIA DE SOFTWARE

OTÁVIO LEMES DE OLIVEIRA

**SISTEMA GERENCIADOR DE BANCO DE DADOS**

Orientado à governança empresarial

Trabalho avaliativo apresentado ao professor Me. Eduardo Ferreira de Souza, como requisito para composição da nota do quarto período de Engenharia de Software.

Orientador(a):Me.Eduardo Ferreira de Souza

OTÁVIO LEMES DE OLIVEIRA

**SISTEMA GERENCIADOR DE BANCO DE DADOS**

Orientado à governança empresarial

Trabalho avaliativo apresentado ao professor Me. Eduardo Ferreira de Souza, como requisito para composição da nota do quarto período de Engenharia de Software.

Prof. Me. Fulano Silva - Orientador

faculdade generica

Prof. Me. Fulano Silva - Orientador

faculdade generica

Prof. Me. Fulano Silva - Orientador

faculdade generica

**SUMÁRIO**

[**1.CONCEITO DE ARQUITETURA DE SOFTWARE**](#_tyjcwt) **4**

[**3.PRINCIPAIS ARQUITETURAS DE SOFTWARE**](#_3dy6vkm) **4**

[**4.ARQUITETURA LAYERS**](#_1t3h5sf) **4**

[**4.1 Design patterns indicados**](#_44sinio) **5**

# 5. *TEMPLATE METHOD* 5

# 

# 

# 

# 

# 1.CONCEITO DE GOVERNANÇA

**2.CONCEITO DE DESIGN PATTERN**

Design pattern consiste em descrever um problema, e sua solução segue uma sequência de passos direcionados para um problema ou rotina específica que já tenha recorrido em outras ocasiões e pode ser replicado no contexto atual em que a sequência estabelecida abrange o arranjo dos objetos e classes para resolver inconsistências ou otimizar rotinas.

# **3.PRINCIPAIS ARQUITETURAS DE SOFTWARE**

As arquiteturas mais populares recebem essa fama por conta de sua estrutura altamente eficientes em solucionar problemas em uma perspectiva abrangente, podemos descrever situações em que a parte de interação com o usuário é mais relevante do que a de nível servidor ou tem a mesma relevância e assim por diante. Diante disso, notam-se como mais relevante as arquiteturas LAYERS, CLIENT-SERVER, MODEL-VIEW-CONTROLLER, MICROSERVICES, PIPES-AND-FILTERS, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE e PUBLISH-SUBSCRIBE.

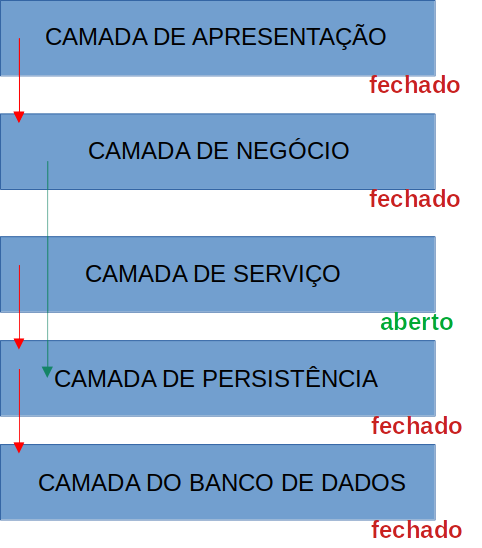
# 4.ARQUITETURA LAYERS

Também conhecida como arquitetura de camadas, ela se divide em camadas distintas que são compostas por funcionalidades, responsabilidade ou regras de negócios relacionadas.

Sua comunicação deve ser explícita e de comunicação estrita e implementação levemente acoplada, uma vez que sua comunicação ocorre tanto entre alto e baixo nível, podendo ser classificada de dois tipos, fechado e aberto.

O fluxo de comunicação fechado se caracteriza pela ordem restrita que o programador estabelece e os dados devem fluir por esse caminho programado sem sofrer desvios. Caso a layer seja aberta, o seu caminho se torna facultativo e a ordem de fluidez de dados pode simplesmente ignorá-la.

Para simplificar o entendimento nota-se o fluxograma abaixo:



É notável que o fluxo estabelecido acontece somente com camadas adjacentes e há divisão rígida nas camadas, portanto evidencia-se que esse tipo de arquitetura é bastante útil em casos que envolvem interação entre responsabilidades de alto e baixo nível.

# 4.1 Design patterns indicados

Trabalhar com arquiteturas requerem boas práticas e uma das melhores maneiras de se obter isso é adotando uma design pattern, um dos melhores modelo para se adotar nesse caso sem dúvida é o padrão “Builder”, pois dentro de suas características se destaca o tratamento dos objetos de maneira complexa, característica essa que muitas vezes as layers demonstram devido o acúmulo de muitos atributos e métodos em si mesmas.

A estrutura do “Builder” é composta por uma classe builder responsável pela definição das etapas de construção de um software, uma classe concrete builder que implementa a interface de acordo com sua necessidade, como se definisse se um layer é aberta ou fechada, a product class, representa a aplicação estruturada e como o fluxo deve acontecer.

# 

# 5. *TEMPLATE METHOD*

Algumas vezes traduzido para o português como “método padrão”, consiste em gerar um código base para algo, e em seguida, permitir que parte desse código seja sobrescrita, permitindo que parte do script original seja aproveitado, ao mesmo tempo que obriga que o padrão seja seguido.  
 Em um paralelo com orientação a objetos, o código base do *Template Method* é como uma função “virtual”, onde é definido o padrão de argumentos e retornos, e os códigos que se baseiam nele são como as funções “*override*”, sobrescrevendo as ações, contudo, seguindo a estrutura, como argumentos e tipo de retorno.

# 

# 

# 

# 

**4.BIBLIOGRAFIA**

##### 