****

**FACULDADE DE TECNOLOGIA RUBENS LARA**

**Alexandre Gomes da Silva**

**Rodrigo Vasconcelos da Silva**

**Portal da vida politica**

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia Rubens Lara, como parte dos requisitos para a obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador: Rosemeire Cardozo Vidal**

Professor

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Santos – SP**

**2013**

Sumário

[Capitulo 1 - INTRODUÇÃO 4](#_Toc372564559)

[1. TEMA E PROBLEMATIZAÇÃO 4](#_Toc372564560)

[Contextualização 4](#_Toc372564561)

[Problema de Pesquisa 4](#_Toc372564562)

[Hipóteses 5](#_Toc372564563)

[Relevância 5](#_Toc372564564)

[2. OBJETIVOS 6](#_Toc372564565)

[Geral 6](#_Toc372564566)

[Específicos 6](#_Toc372564567)

[3. Ferramentas e metodologia 7](#_Toc372564568)

[Scrum 7](#_Toc372564569)

[MVC 7](#_Toc372564570)

[JAVA 7](#_Toc372564571)

[Spring Framework 7](#_Toc372564572)

[*4.* ABORDAGEM *METODOLÓGICA* 8](#_Toc372564573)

[Revisão Bibliográfica 8](#_Toc372564574)

[Quantitativa 9](#_Toc372564575)

[5. CRONOGRAMA 10](#_Toc372564576)

[Capitulo 2- Definições e planejamento 11](#_Toc372564577)

[1. Definição de papeis 11](#_Toc372564578)

[Usuário 11](#_Toc372564579)

[Moderador 11](#_Toc372564580)

[Administrador 12](#_Toc372564581)

[2. Ferramentas e sua utilização no fluxo de desenvolvimento 12](#_Toc372564582)

[Orientação a objetos 12](#_Toc372564583)

[Histórico da Linguagem Java 13](#_Toc372564584)

[Arquitetura MVC 13](#_Toc372564585)

[Eclipse IDE 13](#_Toc372564586)

[Spring Framework 14](#_Toc372564587)

[Scrum 15](#_Toc372564588)

[3. Diagrama de Classe 16](#_Toc372564589)

[4. Diagrama de Caso de Uso 17](#_Toc372564590)

[5. Diagrama de Atividade 18](#_Toc372564591)

[Ação: Envio de mensagem 18](#_Toc372564592)

[Ação: Inserção de novo tópico 19](#_Toc372564593)

[Ação: Interação de Moderador com tópico e comentário 20](#_Toc372564594)

[Ação: Exclusão de usuários 21](#_Toc372564595)

[6. REFERÊNCIAS BIBLIOGRÁFICAS 22](#_Toc372564596)

Capítulo 1 - INTRODUÇÃO

# TEMA E PROBLEMATIZAÇÃO

À medida que as eleições se aproximam as propagandas politicas, principalmente televisivas passam a fazer parte do nosso dia-a-dia, o que possibilita aos eleitores conhecer melhor os candidatos, suas propostas e ideias. Atualmente, segundo estatísticas, grande parte dos eleitores conhece o candidato apenas pelos meios midiáticos tradicionais, uma forma que mostra apenas o que o candidato fez de bom em sua vida pública. Analisando dessa maneira, a fonte de informação que chega até o eleitor é pouco diversificada, consequentemente por estar acostumado a utilizar esta fonte acaba sendo forte influenciado, sendo mostrado a ele apenas informações a favor do candidato ou do partido. Diante do exposto, a internet pode ser uma forte aliada na obtenção de informações reais sobre os candidatos, que por ser uma rede mundial possibilita a circulação de uma série de informações, que poderiam ser centralizadas e filtradas. Um portal colaborativo com uma base de dados, alimentada com informações confiável obtidas pelos próprios eleitores sendo uma alternativa para meios midiáticos tradicionais, composto de evidências e fontes diversificadas, de maneira que o eleitor possa tirar suas próprias conclusões sobre o candidato e posteriormente fazer sua escolha baseado nas informações obtidas.

## Contextualização

Em tempos de manifestações e protestos, cujos objetivos são os mais variados, o eleitor espera do governo, resultados que foram prometidos em tempos de eleição. Desta forma há a necessidade de mais transparência sobre a vida política dos candidatos, para que na hora de decidir o voto, o eleitor esteja ciente em quem realmente esta votando, com um conhecimento baseado em informações verídicas e divulgado por vários outros eleitores, podendo a qualquer momento ter acesso a essas informações de forma fácil, rápida e centralizada.

## Problema de Pesquisa

Hoje temos acesso a uma massa de informação muito grande, disponibilizadas pelos mais variados meios de comunicação, porém quando se fala de politica, a informação se torna essencial para uma escolha consciente do candidato, pois o poder de escolha permite o poder de mudança. Confiar apenas nas informações que nos são trazidas por propagandas televisivas, por exemplo, não garante que essas não tiveram algum tipo de manipulação, seja pessoal ou partidária, principalmente em tempos de eleição onde são apresentadas no horário politico pesquisas de pretensão de votos de origens diversas nem sempre confiáveis, indicando candidato do partido ao cargo, com índices que podem não ser verídicos.

## Hipóteses

Os meios atuais para obtenção de informação na escolha do voto satisfazem a necessidade de informação do eleitor. Se houvesse um espaço de comunicação confiável, onde todos os eleitores pudessem ter acesso a informações do candidato, seu mandato atual ou anteriores, processos em que está sendo julgado, o eleitor poderia se manter atualizado e se conscientizar, sabendo tudo o que o possível eleito em questão, fez e esta fazendo para prover melhorias e demonstrar ser merecedor de confiança e voto nos tempos de eleição.

Hoje há informação de forma a prover ao eleitor a oportunidade de se auto informar sobre os candidatos, com isso, se faz necessário um portal de informações centralizadas, organizada e coerente, para que o eleitor obtenha o conhecimento que necessita antes de votar e durante o mandato do eleito, para assim poder cobrar a confiança que foi depositada no político.

A tecnologia Java junto à arquitetura MVC, *Spring framework* e metodologia *Scrum* supriram tudo o que é necessário para o desenvolvimento, manutenção e evolução de um portal com esse porte. A internet hoje é onde a informação se dissemina de forma sem igual, sem nenhum tipo de manipulação ou interesse cabendo ao eleitor verificar a origem da publicação, portanto o cenário é ideal para centralizar as informações sobre os candidatos, pois dessa forma todos poderão ter acesso às informações sem receio de interferências partidárias ou governamentais. Sendo a informação transmitida por qualquer eleitor que queira divulgar algum fato ou acontecimento sobre candidato ou eleito que ainda não foi exposto, ou exposto de forma incorreta pelas mídias.

## 

## Relevância

A importância de um portal com publicações sobre os candidatos nas eleições traz para o eleitor a opção de conhecer melhor seus candidatos, com isso poderá votar com maior propriedade. O portal permitirá que o eleitor forneça informações que forem de seu conhecimento. Havendo moderadores que terão como principal papel verificar a origem da publicação feita pelo usuario, assim o eleitor terá informações trazidas por eleitores.

# OBJETIVOS

## Geral

O portal terá como medida apenas informar, cabendo ao eleitor formar sua opinião com base nas publicações e noticias do portal, tendo o(s) moderador (es) um cuidado em analisar as matérias disponibilizadas, fazendo uma verdadeira investigação sobre a legitimidade da informação. Com o objetivo de conscientizar o eleitor sobre o(s) candidato(s) que se pretende votar, mostrando suas ações praticadas e omitidas perante o cenário político antes, durante e depois das eleições. O portal será uma importante ferramenta de monitoramento sobre tudo o que venha a ter uma relevância política e seja direito do eleitor saber.

Com a ideia de tornar publico o que não é divulgado nas propagandas políticas, o portal levará informação de eleitores para eleitores tendo como a internet um veículo de comunicação. O portal não terá partido, não terá bandeira, seguirá uma única ideologia, a verdade. A internet possui a liberdade necessária para esse tipo de conteúdo, divulgando informações com relevância, para a formação de opinião dos eleitores, com uma linguagem fácil onde até mesmo os mais leigos em política, poderão compreender. Muitas dessas informações estão postadas em *sites* governamentais, porém com uma linguagem maçante e de difícil entendimento, nada atrativa e dificultando assim os eleitores de se manterem informados.

## Específicos

Este projeto iniciará com a elaboração de documentação funcional para atender as necessidades de controle de um projeto, com foco principal em tarefas, recursos e custos. Posteriormente serão elaborados os descritivos técnicos, modelo de dados e diagramas UML. Terminada a primeira fase de documentação, será iniciada a fase de desenvolvimento do sistema, que engloba metodologia e ferramentas técnicas.

# Ferramentas e metodologia

## Scrum

Uma metodologia de desenvolvimento ágil que gerencia projetos organizacionais onde é difícil planejar a frente, voltada a ciclos de *feedback* que compõem o núcleo da técnica de gerenciamento. Constitui-se da tomada de decisão em níveis prioridade, certeza e operação.

## MVC

Uma arquitetura que auxiliar nas aplicações constituídas de três camadas (Modelo, Visão e Controle) independente entre elas, facilitando alterações e customizações de códigos já existentes podendo representar os dados de diversas formas diferentes.

## JAVA

Java é uma tecnologia orientada a objeto (tipo de paradigma de analise que considera todos os elementos inseridos no sistema objetos) que traz a programação a um nível de linguagem próximo do cotidiano, usada para desenvolvimento de aplicações desktops e web.

# ABORDAGEM METODOLÓGICA

## Revisão Bibliográfica

Toda vez que votamos em um candidato, depositamos nele nossa confiança, imaginamos que este será diferente dos demais, muitas vezes não nos importamos sequer em avaliarmos o passado dele e suas ações mediante ao cenário publico, votamos simplesmente por costume. Em propagandas eleitorais os candidatos impressionam e conquistam o eleitor com planos e projetos de governo que apresentam, porém conforme o tempo passa vemos que as propostas metas são intangíveis e o plano de governo é inconsistente.

É preciso trazer a verdade ao eleitor, verdade esta que na maioria das vezes não é relatadas em campanhas que mostram apenas ‘o bom politico’, deixando de lado processos que este possa estar respondendo, os projetos que participou e que não deram resultado nenhum, os desperdícios de verbas publicas em obras inacabadas ou superfaturadas.

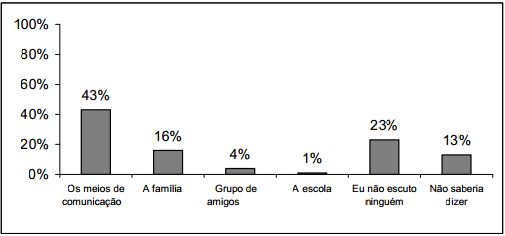
As mídias tradicionais, na grande maioria das vezes, trazem batalhas entre partidos e candidatos, com troca de acusações e informações duvidosas, fazendo com que o eleitor fique ainda mais confuso em sua opção de voto. Buscar informações sobre os candidatos gera algum trabalho, pois na maioria dos casos estas estão muito descentralizadas, e geralmente o eleitor não tem conhecimento de como fazer essa pesquisar.

No inicio o portal será divulgado em vários *links* em redes sociais e foruns que poderá obter mais visibilidade e acessos, trazer informações sobre políticos que aparentemente já são conhecidos pelo eleitor ou que participaram das ultimas eleições. Deverá conter históricos, de projetos com isso despertar a curiosidade dos eleitores sobre determinados políticos, fazendo com que tenham conhecimento em quem terão votar ou em outrora votou, aumentar a curiosidade sobre os demais candidatos podendo compara-los, definindo critérios de voto que antes não tinha possibilidade de efetuar, deixando para traz a forma intuitiva induzida pelas propagandas eleitorais. O resultado é um eleitor mais consciente conhecedor parcial ou até mesmo completo do candidato, com dados confiáveis.

## Quantitativa

Conforme mostra o gráfico os meios de comunicação são onde o eleitor mais busca informações sobre os candidatos. O portal será uma alternativa para o eleitor se informar melhor sobre seus candidatos, confirmar as informações que chegam até ele por propaganda eleitoral, contendo sempre a origem da informação disponível para que o próprio eleitor possa conferi-las. Através do número de acesso poderemos monitorar a aceitação do portal pelos internautas e mostrar o quanto de eleitores estão interessados em conhecer melhor suas opções de voto.

Figura 1



Fonte: Pesquisa sobre o perfil político dos jovens em Londrina (PR), 2006. N= 270.

# CRONOGRAMA

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **2014** | | | | **2013** | | | | |
| **Atividades** | **Mar** | **Abr** | **Mai** | **Jun** | **Jul** | **Ago** | **Set** | **Out** | **Nov** |
| **Pesquisa do tema** |  |  |  |  |  | **2013** | **2013** |  |  |
| **Definição do tema** |  |  |  |  |  |  | **2013** |  |  |
| **Pesquisa Bibliográfica** |  |  |  |  |  |  |  | **2013** | **2013** |
| **Coleta de Dados** | **2014** |  |  |  |  |  |  |  |  |
| **Apresentação e discussão dos dados** |  | **2014** | **2014** |  |  |  |  |  |  |
| **Elaboração do projeto** |  |  | **2014** | **2014** |  |  |  |  |  |
| **Entrega do projeto** |  |  |  |  | **2014** |  |  |  |  |

Capítulo 2- Definições e planejamento

# Definição de papeis

## Usuário

O usuário é o membro (colaborador) do portal, é ele que publicará a maioria das notícias, poderá debater sobre as matérias tendo como regra o respeito mutuo e o linguajar apropriado, respeitando a opinião dos demais membros. O conteúdo antes de publicado será avaliado pelos moderadores para garantir que a informação possui uma fonte confiável.

As informações necessárias para um eleitor consciente. Inicialmente o novo usuário terá direito a disponibilizar noticias, o moderador ira avaliar o numero de acessos em uma faixa de tempo que dirá se o membro terá direito publicação. O usuário poderá sofrer bloqueios temporários de visualização e até ser banido caso o moderador julgar necessário. Existindo a necessidade de um controle de usuário teremos três tipos de usuários, os recém cadastrados, onde terão apenas permissão de leitura ao portal, os usuários que estão cadastrados a um tempo intermediário, onde poderão incluir informações e apenas visualizar os comentários e os usuários que já estão cadastrados a um tempo satisfatório, que terá permissão total as funcionalidades de membro.

## Moderador

Um moderador por definição é alguém que gerencia algo, se tratando de um sistema (portal) o moderador irá controlar o comportamento dos usuários, para isso terá funcionalidades diferenciadas contendo responsabilidades e privilégios, é ele que irá gerir o comportamento dos membros, tendo um papel fundamental na organização e cumprimento das normas do portal.

O moderador poderá a qualquer momento punir os usuários que não tenha um comportamento adequado ou descumpram as regras, tendo como medida penalidades como suspensão por um determinado tempo ou exclusão do portal, funcionalidades essas que o moderador deverá ter para que as políticas de conduta sejam cumpridas pelos membros do portal.

O moderador terá que gerenciar os membros e os conflitos entre eles, supervisionar o que é postado para identificar se existe algo que esta sendo descumprido conforme as regras impostas, identificar publicação de matérias que tenham como fim propaganda eleitoral, editar, mover e apagar matérias descabidas ou fora da ideologia do portal. O moderador terá como perfil uma pessoa equilibrada capaz de manter a ordem apaziguando brigas, procurando manter um bom relacionamento com os usuários, ser ético e ter bom censo em suas ações. É necessário que o moderador tenha disponibilidade (estar on-line) para esclarecer as duvidas que surjam por parte dos membros, não sendo necessário saber todas as respostas, mas sim as regras e normas do portal e nem estar disponível vinte quatro horas por dia.

O cargo de moderado trata-se apenas de um membro do portal com boa vontade que se dispôs a dar um pouco de seu tempo ao portal, sendo um trabalho voluntario onde não se recebe nenhum tipo de compensação e sim responsabilidades tendo ele que cumprir determinadas tarefas do cargo. Para ser qualificado como um moderador o usuário deverá ser membro ativo no portal e como requisito ter um histórico sem punições, procurar ajudar os membros mais novos esclarecendo duvidas e postar novidades. Deverá o moderador conhecer bem as normas do portal para saber analisar as publicações dos membros, para não cometer erros e assim perder a qualificação de moderador.

## Administrador

O administrador (membro da equipe de desenvolvimento) é quem vai controlar o fluxo de vida dos usuários no portal, interagindo apenas com os moderadores, terá as funcionalidades completa do portal podendo criar e excluir um moderador como tornar um membro um moderador e vice-versa. Será disponibilizado pelo administrador os termos de uso, onde será detalhado o que o usuário tem direito, infrações e suas penalidades, o mesmo servindo para moderador que esta sujeito as mesmas penalidades e perda do função.

# Ferramentas e sua utilização no fluxo de desenvolvimento

## Orientação a objetos

As primeiras linguagens de programação que surgiram, tinha o conceito de programação estruturada, paradigma em que o código resume-se a três estruturas básicas, sequência, decisão e iteração. Com a evolução da informática e o surgimento de novas linguagens de programação, criou-se um novo paradigma, a programação orientada a objetos, que consiste na composição e interação entre diversas unidades de software chamadas de objetos. Uma das grandes melhorias que esse paradigma trouxe, foi a capacidade de abstrair o mundo real na estrutura do software, criando objetos que possuem atributos e comportamentos. Segundo o TIOBE *Programming Community* Index um famoso indicador de popularidade pode-se afirmar que atualmente a grande maioria dos softwares desenvolvidos utilizam alguma linguagem com suporte à programação orientada a objetos.

## Histórico da Linguagem Java

Java surgiu na década de 90, oriundo da linguagem *Oak*( que foi criada para o projeto \*7 da Sun), Java foi a adaptação do Oak para internet e foi lançado em 1995 por James Gosling e sua equipe. A linguagem Java foi amplamente adotada após seu lançamento e em 2004 já atingia o número de 3 milhões de desenvolvedores em todo o mundo.

Java se destaca, principalmente, pela sua interoperabilidade, por ser compilada para um byte code e não para um código nativo, como a grande maioria das linguagens, Java pode ser usado em qualquer dispositivo que executa uma máquina virtual Java.

Em 1999 foi criado o Java EE (Java *Enterprise* *Edition*), ele consiste em uma série e especificações que demonstram como deve ser implementado um *software* que utilize uma infraestrutura complexa, composta por: persistência em base de dados, transação, acesso remoto, web *services*, gerenciamento de threads, gerenciamento de conexões HTTP, cache de objetos, balanceamento de carga, entre outros. A ideia principal do Java EE é entregar pronta toda essa infraestrutura para o desenvolvedor, possibilitando assim que o foco seja voltado para a implementação das regras de negócio do software.

## Arquitetura MVC

Para desenvolver uma aplicação, é necessário definir qual a arquitetura será utilizada, existem alguns modelos tradicionais de arquitetura de software, como: cliente-servidor, monolítica, orientada a serviço, orientada a busca e o modelo em três camadas, onde encaixa-se a arquitetura MVC. MVC é o acrônimo de *Model*, *View* e *Controller*(Modelo, visão e controlador). Esta arquitetura visa a separação do código de uma aplicação nessas três camadas, na camada de modelo, temos as informações do domínio que a aplicação opera, um exemplo são as classes POJO (*Plain old java objects*), classes que mapeia todas as entidades com que a aplicação irá interagir. Na camada view são exibidas as informações do domínio para o usuário, por exemplo os dados de um usuário do sistema, que estão mapeados em uma tabela na base de dados são exibidos através de uma página web, o código dessa página faz parte da camada de *view.* A camada *Controller* realiza o processamento e responde aos eventos gerados pela *view,* podemos dizer que ela é uma intermediária entre as outras duas camadas, também é nela que são realizadas validações dos dados e filtros impostos pelo usuário.

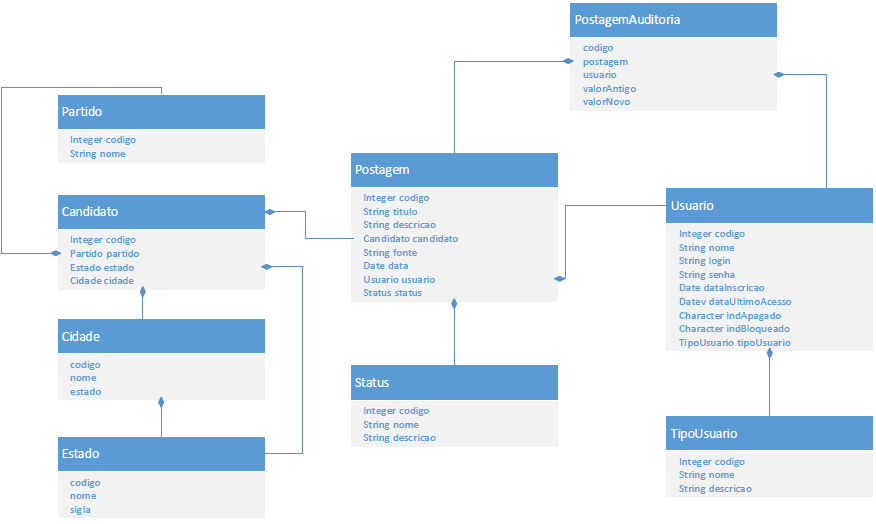
## Eclipse IDE

IDE (*Integrated development environment*) são softwares que auxiliam no desenvolvimento de aplicações, eles possuem uma série de ferramentas que possibilitam e agilizam o desenvolvimento de aplicações, tais como: compilados, edito, linker, depurados, modelagem, geração de código e refatoração. As principais IDE’s que dão suporte a linguagem Java atualmente são: Eclipse, NetBeans e InteliJ. O Eclipse é largamente utilizado, pois além de Java possui suporte a outras linguagens, como PHP, Pearl, Ruby, Groove e outras. Além de possuir uma série de ferramentas padronizadas em suas versões, uma dos grandes diferenciais da IDE Eclipse, são os vários plug-ins que ela possui, para uma infinidade de funções, com essa condição, cada desenvolvedor pode customizar a IDE de acordo com as suas necessidades, o que ajuda tanto em termos de organização, como também no consumo de recursos de hardware.

## Scrum

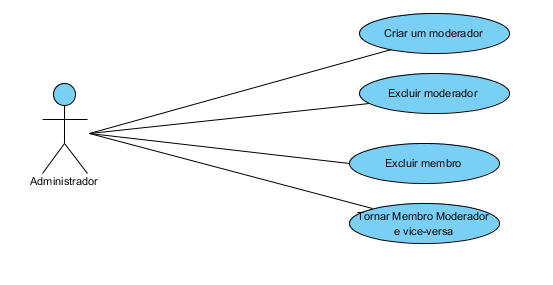
Atualmente existem uma série de metodologias de desenvolvimento de software, como, RUP, XP e outras. Dentre elas existe uma categoria metodologias ágeis, sendo o *Scrum* uma das mais conhecidas e utilizadas. No *Scrum* o processo de desenvolvimento é iterativo e incremental, constituído de ciclos de feedback entre equipe e clientes. Uma das grandes características do *Scrum* é a entrega contínua de funcionalidades, o que permite acompanhar bem de perto a evolução do que está sendo desenvolvido. De forma básica, temos os *sprints*, que são as unidades básicas de desenvolvimento, onde são criados incrementos de produto potencialmente entregável, algo como pequenos protótipos. Antes de iniciar cada *sprint*, há um *sprint planning*, que basicamente é uma reunião que vai definir os itens que devem ser entregues ao final desse *spring,* esses itens fazem parte do *Sprint backlog*, uma lista de atividades que devem ser concluídas nesse spring. Durante os *sprints*, temos as stand-up meetings, reuniões diárias onde a equipe conversa sobre o trabalho que está sendo desenvolvido.

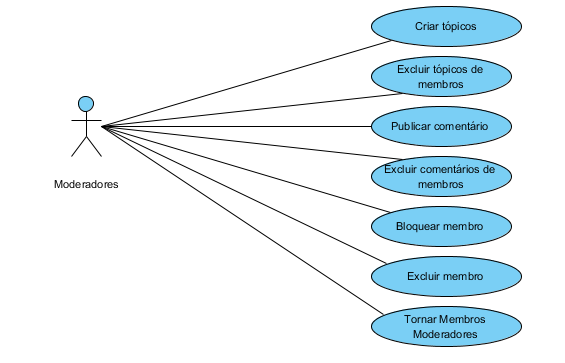
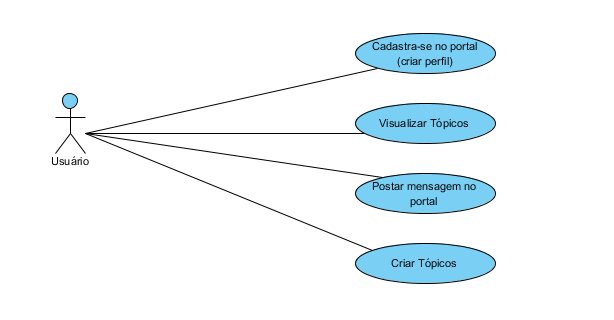
# Diagrama de Classe

******

# 

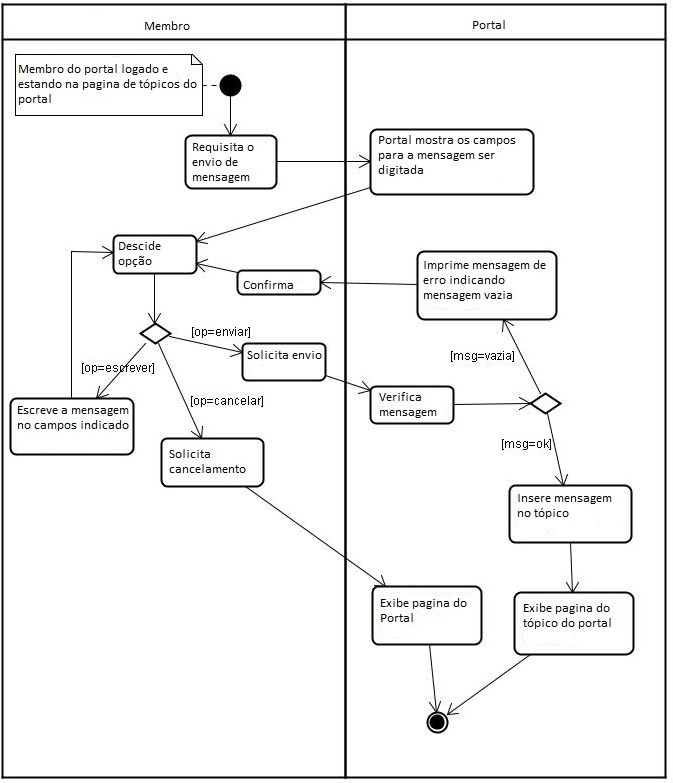
# Diagrama de Caso de Uso



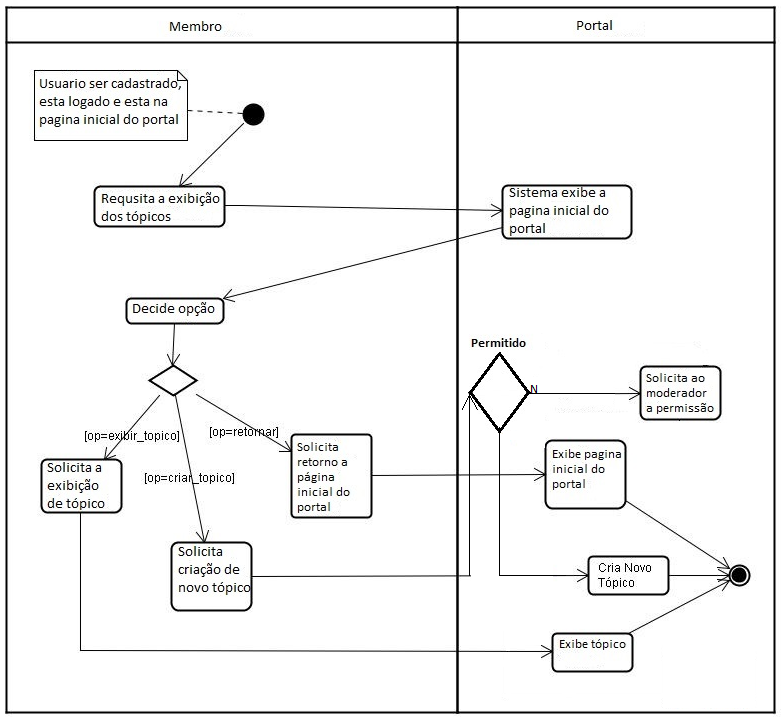


# Diagrama de Atividade

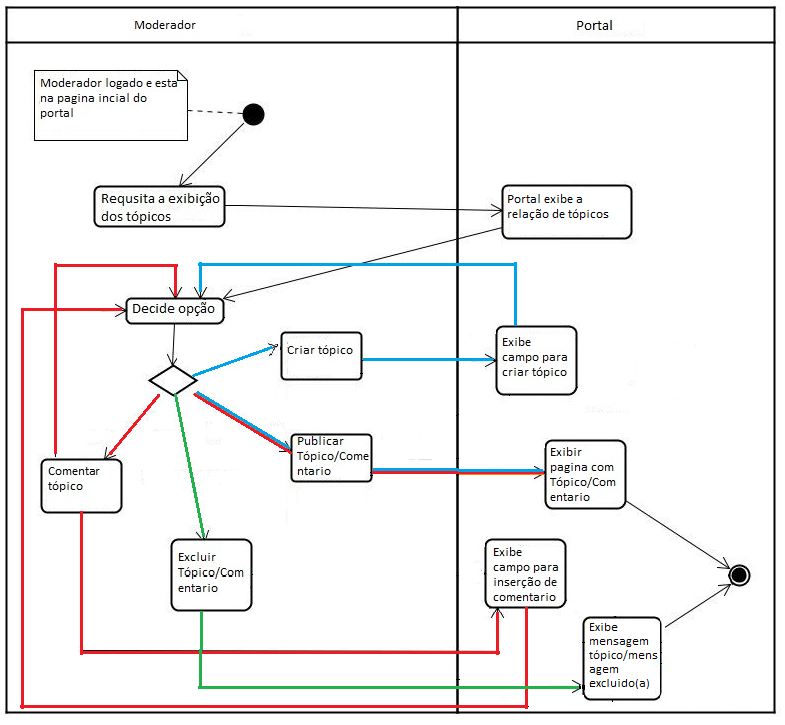
## Ação: Envio de mensagem



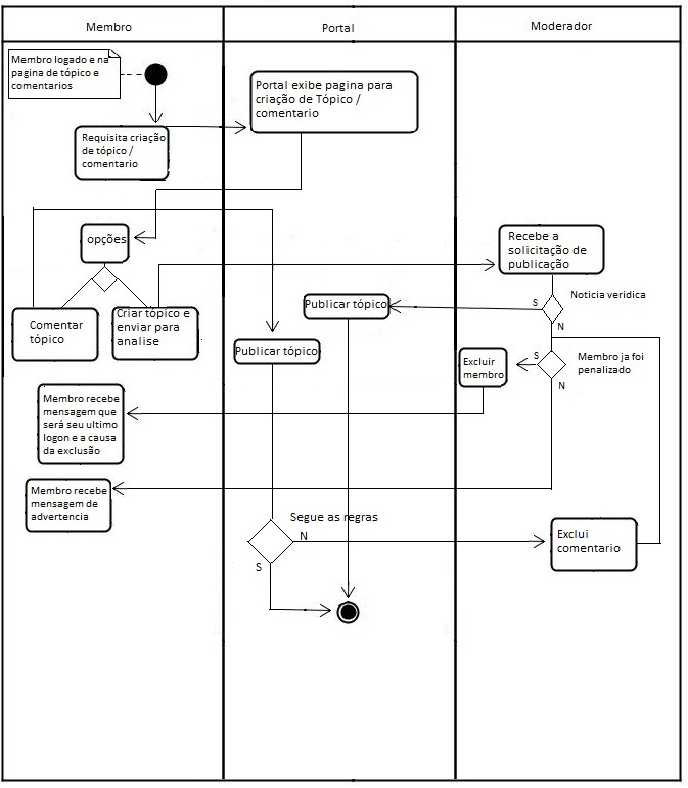
## Ação: Inserção de novo tópico



## Ação: Interação de Moderador com tópico e comentário



## Ação: Exclusão de usuários



Capítulo 3- Conceitos Programáticos de Desenvolvimento

Objetivo

O objetivo deste capítulo é conhecer os conceitos das diferentes tecnologias de mercado que serão utilizadas no projeto, quais suas vantagens e desvantagens. A melhor maneira de se escolher uma com uma ferramenta é conhecendo o suas necessidades, com isso é possível escolher a que melhor se encaixa no projeto.

Segundo Csillag (1995), “Uma ferramenta utilizada para reduzir custos de produção de bens e serviços e aumentar o valor para o usuário. Consiste basicamente em identificar as funções de determinado produto avalia-las e finalmente propor uma forma alternativa de desempenha-las de maneira mais conveniente do que a conhecida”.

Com as atuais tecnologias de mercado, o desenvolvimento de software está cada vez mais automatizado, tornando os esforços de criação e manutenção cada vez menor, com isso varias dificuldades que os desenvolvedores enfrentavam no passado, hoje vão sendo superadas pelas novas tecnologias que surgem todo dia no mercado.

As implementações de conceitos criaram tecnologias que vão desde tratamento de acesso e armazenamento de dados, até layouts sofisticados e modernos. Com todo esse aparato tecnológico no mercado é necessário ter claro entendimento do conceito das tecnologias antes de implementa-las para podermos identificar se está alinhada com o desenvolvimento e se é uma solução viável para a necessidade do software.

Fazendo analogia a um prédio, para construirmos um software é necessário começarmos pela base, havendo uma base bem estruturada com bons alicerces conseguimos erguer um prédio com seus andares, assim é a base de dados de um software precisa ser criada de forma que possa ser expandida, de fácil manutenção e volatilidade nas mudanças de requisitos de negocio.

De inicio apresentaremos um conceito que será utilizado para o mapeamento de classes. Diante das diversas dificuldades que envolvem os ciclos de vida de software um dos problemas mais propensos a falhas e que traz muitas dificuldades para os desenvolvedores e analistas é a evolução e a definição de acesso à base de dados para aplicações orientadas a objeto, pensando nisso é necessário uma maneira de mapear a base de dados relacional para a linguagem de programação que no caso é orientado a objeto.

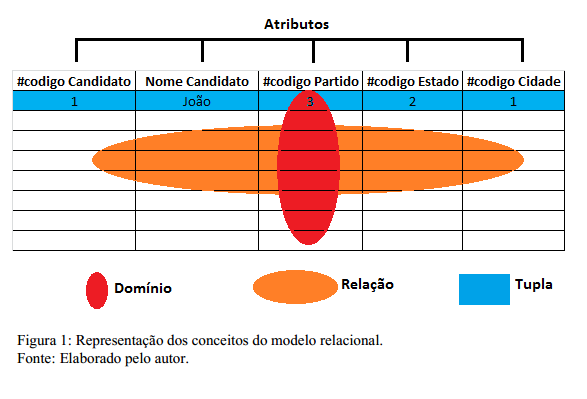
Existe um conceito chamado de mapeamento objeto-relacional, mas para entender claramente o seu conceito é preciso ter uma definição dos conceitos de banco de dados relacional e o paradigma de orientação objeto.

-Banco de dados Relacional

Obter informações de forma ágil e confiável é vital para as empresas de atualmente em dia, com isso se faz necessário um banco de dados confiável que já esteja consolidado no mercado, como o banco de dados relacional. O banco de dados relacional é caracterizado por organizar os dados em relações, relações que podem ser considerada tabelas, colunas como atributos da relação, as linhas representam as *tuplas* ou elementos da relação e o domínio é os tipo de dados que descrevem os tipos de valores que pode ser representado em cada coluna. Segundo Elmasri e Navathe (2005), o modelo relacional representa o banco de dados como uma coleção de relações

Um conceito muito importante em banco de dados relacional são os atributos chaves, é com isso que ele pode diferenciar as *tuplas* uma das outras e manter relações entre múltiplas tabelas, também existe a possibilidade de acelerar o acesso às consultas usando índices.

Segundo Kaufeld (1996), o modelo de banco de dados relacional possui a capacidade de lidar com grandes volumes de informações, eliminando dados redundantes.



Um banco de dados relacional é composto por inúmeras tabelas que diariamente são alimentadas por sistemas em funcionamento, uma massa de dados que a empresa manuseia para obter informações como estáticas e relatórios, dependendo do numero de informações que se deseja obter uma empresa pode ter vários bancos de dados em diferentes domínios de aplicações.

Chaves Primarias e estrangeiras

Um banco de dados relacional caracteriza-se pela utilização de chaves sendo os tipos identidade e referencial, é isso que mantem a restrição de integridade, havendo entre as tabelas relacionamentos através de um campo identificador como chave. Uma chave primaria pode ser representada por uma ou mais colunas, sendo que sua principal característica é que ela não pode ter valores duplicados dentro de uma tabela. Uma chave estrangeira pode ser representada por uma ou mais colunas ela se se caracteriza por ter o mesmo valor da chave primaria de outra tabela, sendo a chave estrangeira que definir o relacionamento entre as tabelas. Uma chave primaria é escolhida a partir de um conjunto de candidatas de uma entidade que por definição tenha valor exclusivo das tuplas e não nulo.

Exemplo:

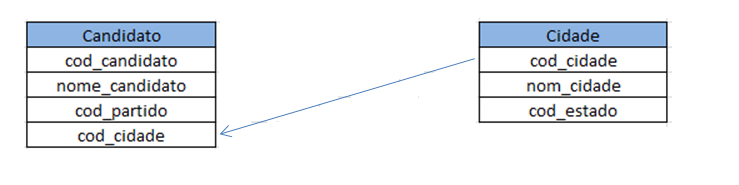


Figura 2: Representação dos conceitos de chave estrangeira

Fonte: Elaborado pelo autor.

Em duas entidades uma chamada Candidatos que contem a chave primaria “*cod\_candidato*” e as chaves estrangeiras “*cod\_partido*” e ”*cod\_cidade*”, já a outra entidade contem a chave primaria “*cod\_cidade*” e a chave estrangeira “*cod\_estado*”, nessa situação a chave estrangeira “*cod\_cidade*” da entidade Candidato contem o mesmo valor da chave primaria da entidade Cidade formando assim um relacionamento entre as duas entidades, o mesmo se aplica para as chaves estrangeiras “*cod\_partido*” da entidade Candidato e a “*cod\_estado*” da entidade Cidade.

Integridade

O conceito de integridade diz respeito à credibilidade de uma ou mais informações em banco de dados. Ela é necessária para que o banco possa tratar os diferentes tipos de dados de forma especifica minimizando assim a probabilidade de erro. As regras de restrição de banco de dados é que determina que tipo de dados possa ser armazenado em determinada coluna de uma tabela.

Integridade de Entidade

É especificada em relações individuais, nenhum valor de chave primaria pode ser nulo, pois sendo nulo não saberíamos distingui-las. Um exemplo é se uma ou mais *tuplas* tivesse valores nulos em suas chaves primarias não saberíamos distingui-las umas das outras.

Integridade Referencial

A restrição de integridade referencial é especificada entre duas relações e é utilizada para manter a consistência entre *tuplas* destas relações. Informalmente, a restrição de integridade referencial declara que uma *tupla* em uma relação que se refere à outra relação deve se referir a uma *tupla* existente naquela relação (Elmasri e Navathe, 2005).

-Conceito de orientação a objetos

É baseado na composição e interação entre diversas unidades do software chamados de objeto, que possuem responsabilidades e funções e ao ser implementado tenta modelar os elementos no sistema na mesma forma que são percebidos no mundo real.

Segundo Campos (2009), o paradigma de orientação a objeto tenta desenhar os elementos em um sistema da mesma maneira que são percebidos no mundo real, Nesta forma os objetos possuem responsabilidades e funções para atingirem um objetivo comum. Como vantagens a orientação objeto é mais fácil pensar do que em procedures e funções, pois os objetos estão mais próximos do mundo real e o reaproveitamento do código com componentes bem modularizados com funções bem definidas servindo a propósitos claros e delimitados, podendo serem reaproveitados minimizando o tamanho do código.

Seguindo os exemplos voltados ao projeto segue representação de um objeto:

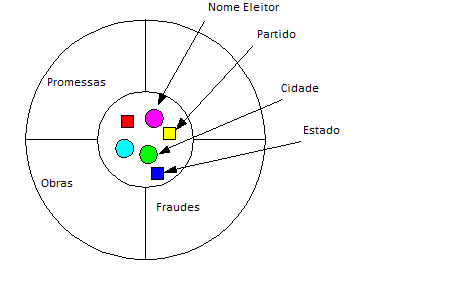


Figura 3: Representação do conceito Objeto

Fonte: Elaborado pelo autor.

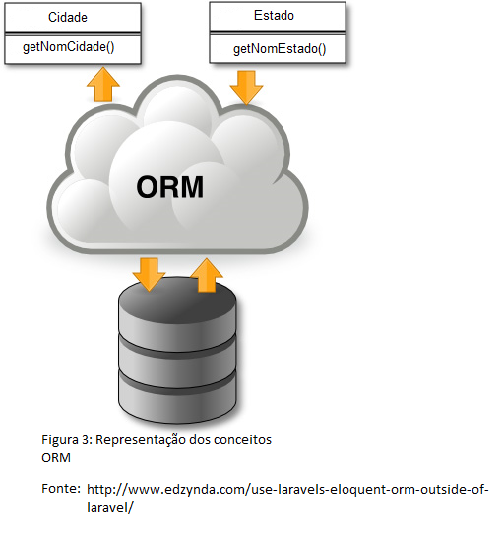
Nesse exemplo temos dois círculos um dentro do outro, o circulo interno representa os atributos, o circulo externo representa as ações ou métodos do objeto.

-Mapeamento Objeto-relacional

Com Orientação objeto e banco de dados relacional definidos, podemos entender o conceito de mapeamento objeto-relacional. Segundo Bauer e King (2005), Havendo explicitamente uma incompatibilidade na relação de tabelas do banco de dados relacional e na rede de objetos em uma linguagem de programação orientada a objeto, passou-se a questionar a importância e o custo que isso poria trazer. O mapeamento objeto relacional (ORM) com o intuito de atingir as vantagens competitivas durante o desenvolvimento de software e evitando o trabalhoso acesso aos dados da forma tradicional, caracteriza-se por ser uma técnica que abstrai a solução de uma forma que as relações passam a ser representadas pelas classes, sendo as “*tuplas”* desta tabela originadas de cada instancia da classe em questão e utilizando banco de dados relacionais reduzindo os impedimentos da programação orientada a objeto.

Softwares desenvolvidos com ORM tendem a serem econômicos no seu desenvolvimento, melhores no desempenho e fácil manutenção e evolução.

Exemplo de abstração de Objeto-relacional.



No mercado existem vários produtos que são utilizados hoje que usam o conceito de ORM que *são Toplink, Ibatis* *e Hibernate*.

# Padrões de Projeto

Introdução

Uma forma de obter qualidade e custos reduzidos em projetos em prazos reduzidos é adotando padrões em seu desenvolvimento, tendo isso em vista se faz necessário conhecer os conceitos e definições dos padrões de projetos para saber qual, quando e onde aplicá-los, sabendo que o seu uso não gerenciado pode torna uma solução em um transtorno para o desenvolvedor.

Segundo José Carlos Macoratti, “Padrões de projetos são soluções para problemas que alguém um dia teve e resolveu aplicando um modelo que foi documentado e que você pode adaptar integralmente ou de acordo com necessidade de sua solução”.

Atualmente softwares com desenvolvimento voltado a orientação objeto vem se destacando no mercado, por agregar qualidades importantes para os sistemas desenvolvidos, como reaproveitamento do código e a capacidade de ser extensível, mas apenas utilizar orientação objeto não garante que essas qualidades sejam obtidas.

Analisando o cotidiano de desenvolvimento de software é possível identificar que muitas soluções para problemas específicos que foram identificados em projetos anteriores são semelhantes ou até iguais aos de novos projetos, mas por deficiência no processo de documentação ou compreensão do problema, não são aplicados nos novos projetos gerando uma perda de tempo e recursos com o desenvolvimento de uma nova solução para um problema que anteriormente já havia sido encontrado ou resolvido. Uma boa pratica que os projetistas costumam adotam é não resolver os problemas do zero, buscando reutilizar soluções já aplicadas em problemas anteriores. Então podemos dizer que padrões de projeto são soluções de problemas que alguém enfrentou e resolveu, aplicando a partir de um modelo documentado e que pode ser adaptado dependendo da necessidade de sua solução.

Histórico

Segundo Christopher Alexander, “cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca faze- lo da mesma maneira”.

É importante ter uma definição clara do problema e conhecer o padrão de projeto que melhor se encaixa no problema, sabendo qual usar, como usar, porque usar e analisar se ele realmente se enquadra no caso em foco.

A principio a iniciativa de projetar soluções a partir de algo conhecido e documentado não partiu de projetos de software e sim da arquitetura civil com Christopher Alexander, quando o mesmo publicou uma catalogo com mais de 250 soluções de projetos de arquitetura. Em seu catalogo foram descritos os problemas e justificados as soluções comuns enfrentados pelos arquitetos no cotidiano.

Sua analise consistia em diminuir o foco do problema procurando estruturas que resolvam problemas similares, assim ele podia entender as similaridades entre projetos distintos, similaridades essas que ele nomeou como padrões. Com o tempo a comunidade de software aderiu a suas ideias que obteve atenção na conferencia sobre orientação a objetos, em 1995 Erich Gama, Richard Helm, Ralph Johnson, John Vlissides publicaram o livro Design Patterns G4 que ganhou sua versão na língua portuguesa chamada “Padrões de Projeto”- Soluções reutilizáveis de software orientado a objetos.

Sendo um catalogo com 23 padrões de projetos para soluções de problema de software contendo três categorias padrões de criação, estruturais e comportamento.

Tipos de Padrões de Projeto.

Segundo Fowler (2006, p 31), sempre que se implanta padrões é necessário adaptar-se conforme o projeto. Observa-se a mesma solução diversas vezes, mas ela nunca é a mesma.

Com a definição do que é padrão de projetos, será apresentado os tipos de padrões de projetos que utilizaremos em nosso sistema, mostrando seu conceito e posteriormente suas implementações e ferramentas, trazendo as características de cada padrão e mostrando de que forma ele será útil para o desenvolvimento do software.

DAO

Um dos problemas enfrentado por desenvolvedores é colocar no código do sistema o *Structured Query Language* (SQL) de forma que não haja duplicidade nos comandos que possuem a mesma finalidade ou comando SQL iguais em lugares diferentes atendendo dois objetos diferentes. A situações em que uma série de comandos *sql* estão espalhados e desordenados tornando o código depreciado, de difícil manutenibilidade e evolução para os desenvolvedores.(impede um tipo de falha de segurança chamada sql injection falha)

A melhor pratica quando se desenvolve um sistema com banco de dados é definir de forma eficaz e organizada onde agrupar os comandos *sql*. A partir disso uma solução seria agrupar todo o código *sql* em um objeto para que a sua manutenção permaneça em um único ponto, é essa solução que se baseia o padrão de projeto de persistência de dados *Data Access Object* (DAO). Segundo Aéce (2005), isso leva a uma produtividade muito grande e facilidades em manutenções futuras.

Genericamente todo o sistema (conjunto de objetos) é composto por relações com a base de dados que são Salvar, Atualizar, Apagar e Listar todos se for o caso, sabendo disso e seguindo a solução proposta pelo DAO fazemos o agrupamento dos comando SQL em classes DAO de acordo com cada objeto, assim as Classes DAO serão o ponto único de manutenção dos comandos *sql.* Segundo, Aéce (2005), é um padrão de projeto que abstrai e encapsula os mecanismos de acesso aos dados.

Segundo Sun (2007), o DAO esconde completamente os detalhes da execução da origem dos dados de seus clientes, conforme mostra.

# 

Figura 4 - Representação do conceito do padrão de projeto DAO.

Fonte: <http://spiproductsplatformsolution.wordpress.com/2012/10/30/hibernate-generic-data-access-object/>

A grande vantagem em desenvolver com o padrão de projeto DAO é que há uma separação rigorosa entre as partes de persistência e regras de negocio, podendo assim evoluir frequentemente independente uma da outra. Quando se altera a logica da regra de negocio, espera-se apenas que se implemente uma interface e quando altera-se a regra de persistência, não se altera a logica da regra de negocio desde que a interface entre ambas não seja modificada.

Exemplo de Classe do Projeto Implementado DAO

public class UsuarioDAO extends DAOGenerico<Usuario>{

public UsuarioDAO(){

super();

}

public void inserirUsuario(final Usuario usuario){

this.inserir(usuario);

}

public void atualizarUsuario(final Usuario usuario){

this.atualizar(usuario);

}

public void excluirUsuario(final Usuario usuario){

this.excluir(usuario);

}

public List<Usuario> listarUsuarios(){

return this.findAll();

}

public Usuario pesquisaUsuarioPorId(final Integer id){

return this.findById(id);

}

public Usuario getUsuarioPorLogin(final String login){

final String sql = "SELECT u FROM Usuario u WHERE u.login = :login)";

final Query q = this.getEntityManager().createQuery(sql);

q.setParameter("login", login);

try{

return (Usuario) q.getSingleResult();

} catch (final NonUniqueResultException e){

return null;

} catch (final NoResultException e){

return null;

} catch(final IllegalStateException e){

return null;

} catch(final QueryTimeoutException e){

return null;

}

}}

Singleton

A técnica de aplicar soluções reutilizáveis em softwares através de padrões de projeto resulta objetos flexíveis, sustentáveis, redução na complexidade do processo de design e ganho de tempo.

Em programação existem dois escopos de variáveis local e global, variáveis locais tem um mecanismo de escopo local de memoria não compartilhada e alocam apenas recursos locais, diferentemente variáveis globais que possuem um escopo acessível a todo um programa, seu conteúdo pode ser potencialmente modificado de qualquer local, e qualquer parte de um código pode depender dela trazendo desvantagens em seu uso, consumindo por vez diversos recursos que não estão sendo necessariamente utilizados, da mesma forma pode-se considerar um objeto de conexão de base de dados, instanciando varias conexões para vários objetos.

O padrão de projeto *Singleton* é uma solução para esse tipo de problema tendo como definição garantir que uma classe tenha somente uma instância e forneça um único ponto global de acesso para a mesma, com isso não há o risco de haver mais de uma instancia para um único objeto, então a classe gerencia sua própria instancia além de proibir que qualquer outra classe crie uma instancia dela, tendo que passar pela classe que gerou o objeto obrigatoriamente. A necessidade de se implementar *singleton* geralmente é quando é preciso globalizar certos aspectos do sistema.

Vantagem

A vantagem que o padrão traz ao ser utilizado é quando existe necessidade de criar uma única instancia do objeto e ao mesmo tempo compartilhar seus recursos com os demais.

Exemplo de Classe do Projeto Implementado *Singleton*

*Delegate*

O padrão de projeto *delegate* é um tipo de variável que armazena o endereço de um método. Sua utilidade se baseia no uso de eventos, quando uma classe dispara uma mensagem em tempo de execução, o *delegate* tem o papel de armazenar o endereço de memoria onde está o método.

Existe a necessidade de alguns requisitos para uma variável apontar para um método com assinatura declarada no *delegate*.

* Uma instancia desta Classe;
* A mesma assinatura do método, sendo que dever ser do tipo *void* e não receber nenhum parâmetro;
* O método devera ser declarado como *static*.

Contendo uma interface publica, as classes do *delegate* permitem inicializar, adicionar, remover e invocar *delegates*.

Existem duas maneiras para invocarmos um *delegate*.

* () Operador da chamada;
* Método *Invoke*.

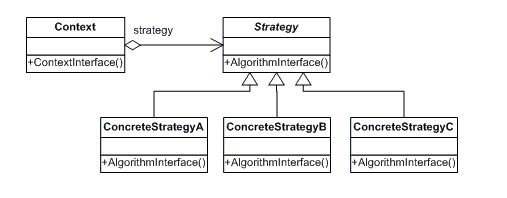
Sintaxe:

Acesso delegate Retorno NomeDelegate()

Strategy

Em sistemas de ambientes corporativos existem lógicas condicionais que são estruturas complexas que tender a se expandir ,e com o tempo se tornar cada vez mais sofisticadas com difícil manuseio. *Strategy* é uma solução que auxiliar a gestão da complexidade que existe na lógica condicional, sugerindo que se desenvolva uma família de classe para cada variação do código e que se forneça para cada classe hospedeira uma instancia de *stategy* para a qual ela delegará em tempo de execução, assim *strategy* nos permite configurar uma classe com um de vários comportamentos, utilizando o conceito de OO chamado de composição.

Exemplo:



**Strategy** - Interface comum para todas as classes (variações concretas) que definem os diversos comportamentos esperados;

**ConcreteStrategy** - Classes que implementam os algoritmos que devem atender a cada contexto;

**Context** - Classe onde os objetos ConcreteStrategy serão instanciados;

**Vantagens**

Substituição de um algoritmo por outro em tempo de execução.

Escolha de diferentes implementações do mesmo comportamento;

Simplificação na classe no fluxo variações de um algoritmo para uma hierarquia;

-Reaproveitamento do Contexto que permite escolher entre uma família de algoritmos que possuem funcionalidades em comum;

Os algoritmos em classes Strategy possuem variações do seus algoritmos independentemente do seu contexto, assim é mais fácil utilizá-los, trocá-los, compreende-los e estende-los;

Minimização ou eliminação da lógica condicional limpando ainda mais os algoritmos;

Exemplo de Classe do Projeto Implementado *Strategy*

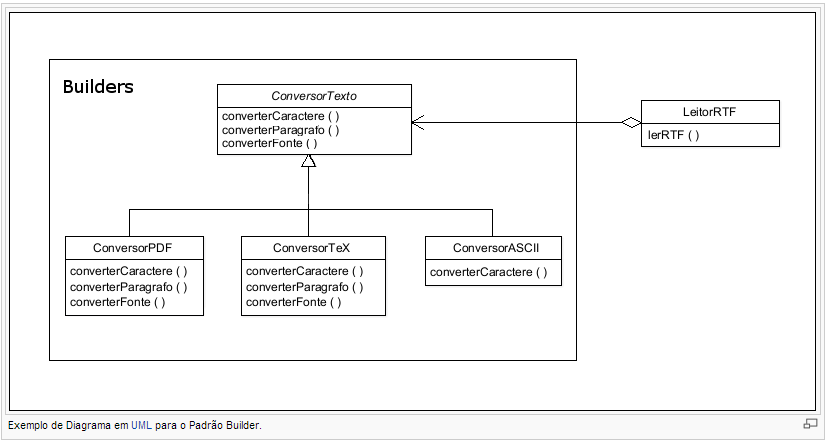
Builder

O Padrão de Projeto Builder é a separação da construção de objetos complexos da sua representação, abstraindo a criação de objetos com particularidades na sua interface, favorecendo a manutenção, uma arquitetura integra e minimizando a complexidade do código. Um objeto de torna complexo quando sua instancia necessita de vários métodos para ser feita, não sendo eficiente ter toda lógica de criação em seu construtor, e nem distribuir a lógica em vários métodos adicionais. Com builder é possível separar a construção do objeto de sua representação podendo assim construir o objeto por etapas.

Vantagem

A Construção de objetos complexos em simples passos, tornando o processo de criação genérico para poder criar diferentes representações.

Exemplo



Fonte: <http://pt.wikipedia.org/wiki/Builder>

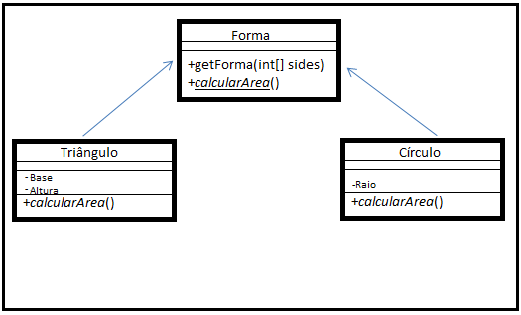
Abstract Factory

O conceito do padrão de projeto Abstract factory é o de utilizar uma interface genérica de determinado objeto, sem especificar a classe concreta do mesmo, assim sua criação se torna dinâmica, sendo desconhecida a classe de implementação, apenas sua interface.

Segundo GAMMA, Definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O Factory Method permite adiar a instanciação para subclasses.

Seu objetivo está em varias classes executarem o mesmo tipo de operação ou método, retornarem o mesmo tipo abstrato mas, internamente instanciar classes diferentes das que o implementam.

Exemplo

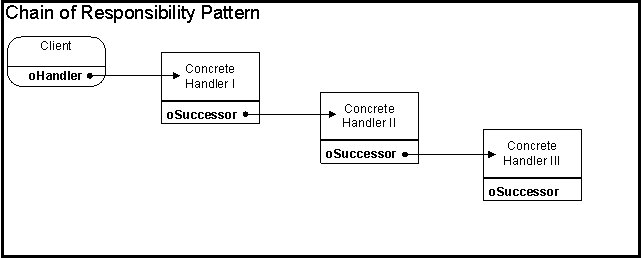


Fonte: Autor

Chain of Responsibility

Com a idéia de evitar o acoplamento entre remetente de uma requisição e o receptor, o padrão de projeto *Chain of Responsability* constrói uma cadeia de objetos receptores e passa a requisição entre eles até que algum receptor tenha uma lógica compatível para trata-lo, assim ele elimina as estruturas de decisão do código. Uma vez que se tem essa estrutura, minimiza-se a dependência do remetente a um só receptor, dando oportunidade para mais de um objeto receptor tratar a requisição do remetente.

Exemplo



Fonte: <http://weblogs.foxite.com/andykramek/2006/12/24/design-patterns-the-chain-of-responsibility/>

Capítulo 4 - Arquitetura de Software

História

O conceito de arquitetura de software tem suas origens relacionadas a projetos de pesquisa realizados pelo cientista holandês *Edsger Dijkstra* em 1968 e o americano *David Parnas* em 1970. Em seus projetos foram destacados a importância da estrutura de um sistema software e o quanto a identificação das mesmas é critica para um sistema. A partir dai no inicio dos anos noventa muitas instituições de pesquisa como *Carnegie Mellon University* e a *University of California* realizavam pesquisas no campo da arquitetura de software, obtendo muita popularidade com estudos de padrões de estilo de arquitetura, documentação de arquitetura, descrição de arquitetura.

Introdução

. A complexidade da informação é um problema que acompanha o campo da computação desde sua criação. A partir disso começou-se a ser desenvolvidas formas de se particionar os sistemas tendo em vista uma compreensão mais precisada software, reuso funções ou partes do código, melhor gerenciamento em execução para minimizando suas complexidades com isso o conceito de arquitetura de software começou a ganhar espaço.

O termo também pode ser usado na documentação de arquitetura de software, tendo como papel o mapeamento dos requisitos iniciais do projeto em alto nível, o reaproveitamento de padrões e componentes entre projetos e a facilitação da comunicação entre os *stakeholders.*

Um dos conceitos de arquitetura de software trata-se da identificação dos componentes de software, suas propriedades externas e sua interação com os demais sistemas. A arquitetura de software especifica de forma abstrata os componentes e suas relações, tendo como objetivo visualizar e documentar o funcionamento da estrutura separadamente da linguagem de programação. Soluções como estrutura de dados, implementação de conceitos de separação de escopo e o desenvolvimento de algoritmos foram aplicados para os primeiros problemas complexos que surgiram, tendo como foco minimizar a complexidade com abstração e separação de interesses. Os componentes de software são elementos que encapsulam uma ou mais funcionalidade, independente e são utilizados para formar sistemas complexos.

A arquitetura pode ser descrita por diversos diagramas proporcionando visões distintas sendo estruturas logicas ou físicas estática ou dinâmica de seu funcionamento. As estruturas logicas englobam conceitos lógicos como classes, funções, variáveis, estruturas físicas corresponde a conjunto de arquivos fonte, bibliotecas, arquivos fontes executáveis que formam a estrutura física do software. Visões Em arquitetura de software existem visões, que são os tipos de arquitetura que são aplicadas em um projeto.

Exemplos:

* Propriedades externas visíveis são exemplos de objetos, classes.
* Configurações de desempenho (rede, processador), módulos de tratamento de rede e uso rede funções para recursos compartilhados.

**Padrões no desenvolvimento de software e Design Arquitetural**

A partir de experiencias colaborativas de programadores, as soluções para problemas similares e recorrentes foram à base dos Padrões.

Os problemas não são isolados contendo forças (propriedades, restrições, requisitos) externas que estão associados aos problemas. A solução obedece às forças do problema, suas interconexões configuração e estrutura.

Os padrões podem ter desde nivel conceitual, a nivel de arquitetura de software, sendo abstratos apresentão soluções de desenvolvimento, podendo ser instanciados quando encotram problemas unicos e similares em diferentes projetos.

Os metodos são ações que o objeto faz ao longo do sistema sendo independentes do problema, assim os metodos não abordam como resolver certos problemas, mas sim, mostram os passos do desenvolvimento e as notações para descrição do sistema, diferentemente dos padrões que são dependentes do problema.

**Categorias de padrões e Relacionamentos**

Cada padrão depende de padrões menores que ele contém e de padrões maiores no qual ele está contido.

Padrões arquiteturais

- expressam o esquema de organização estrutural fundamental para um sistema de software.

Assemelham-se aos Estilos Arquiteturais descritos por [Shaw & Garlan 96].

Padrões de design

- provê um esquema para refinamento dos subsistemas ou componentes de um sistema de software.

Idiomas -

são padrões de baixo nível, específicos para o desenvolvimento em uma determinada linguagem de programação, descrevendo como implementar aspectos particulares de cada componente.

### Divisão de Camadas

Aplicações monolíticas

Antigamente um software era produzido para ser utilizado em uma única maquina, com isso sua criação se continha em um único modulo robusto, que continha todas as funcionalidades pertinentes, gerando uma enorme quantidade de linhas de código, contendo a entrada do usuário, verificação, lógica de negócio e manipulação do banco de dados presentes em um único lugar, tornando a manutenção do software uma tarefa árdua.

Exemplo: Monolíticas



FONTE: <http://www.macoratti.net/vbn_mvc.htm>

Aplicações de duas camadas

Com a necessidade do compartilhamento de informações entre vários usuários simultaneamente, fez surgir à necessidade da arquitetura de duas camadas. Nessa arquitetura o banco de dados não está alocado mais localmente na estação de cada usuário, mas sim em uma maquina especifica, separada das maquinas que executam o software, sendo acessada por vários usuários simultaneamente ficando na estação a Logica de apresentação e a Logica de negocio (clientes ricos). Entretanto surgi um novo problema, que é a gestão de versões do software, para cada atualização que o software vier a ter será necessário atualizar todas as maquinas cliente.

Exemplo: duas camadas



FONTE: <http://www.macoratti.net/vbn_mvc.htm>

Aplicações de três camadas

Com a evolução da internet surgiu à ideia de separar a regra de negocio da interface com o usuário, com a intenção de que os usuários da Web acessem a aplicação sem a necessidade de instalar a aplicação em sua(s) maquina(s). Com essa arquitetura os aplicativos passaram para um Servidor Web podendo ser acessado pelos clientes (cliente pobre ), que passaram a ser os navegadores web.

Exemplo: três camadas



FONTE: <http://www.macoratti.net/vbn_mvc.htm>

Nesse modelo existe a separação Logica e física dos componentes de apresentação (layout) da regra de Negocio, que se encontra no servidor Web e a separação física do acesso aos dados. A separação logica do aplicativo em camadas, á torna flexível permitindo alterações de forma independente. Havendo a divisão da regra de negocio em varias classes reduzindo a dependência entre as classes e pacotes, sendo agrupadas em pacotes é possível reaproveitamento maior do código que pode ser reutilizado em diferentes lugares da aplicação e até mesmo em aplicações distintas.

O modelo de três camadas se torna um padrão no desenvolvimento de softwares corporativos com base na Web. Utilizando uma modelagem baseado na orientação a objeto, nos auxiliará na modularidade, com o encapsulamento das propriedades e estados do objeto e utilizaremos de suas funcionalidades através dos seus métodos, sendo aplicado da maneira corretas a dependência entre os objetos são reduzidas diminuindo o acoplamento e melhorando o código em manutenção e evolução.

MVC

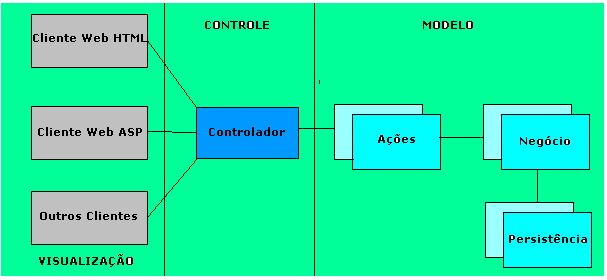
Com as complexidades existentes em desenvolvimento de sistemas utilizando linguagem orientada a objetos, foi necessário o desenvolvimento de técnicas que organizasse melhor o código para melhor compreensão e manutenção do código, sendo feito o mapeamento e separação de componentes, como as tarefas de entrada, processamento, e saída da aplicação; facilitando o mapeamento dos componentes em aplicações Web multicamadas. No modelo de arquitetura MVC a aplicação é separada em três camadas, camada de apresentação, camada intermediaria/negocio e a camada modelo que terá acesso aos dados, sendo atribuído funcionalidades especificas para cada uma das camadas.

A camada modelo representa os acesso e manipulação aos dados e a regra de negocio, que gerenciam a modificação dos dados, mantendo o estado persistente do negocio e fornecendo funcionalidades encapsuladas camada de controle.

A camada de representação exibe ao usuário uma parte em particular do conteúdo da camada modelo e envia para o controlador as ações do usuário, também acessa os dados da camada modelo através de um controlador e define como os dados serão exibidos ao usuário.

A camada de Controle é a que define o comportamento do software, através dela existe a interpretação das ações do usuário para as chamadas da modelo. Em um cliente de aplicação web ações podem serem definidas como o clique de botões, seleções de menus etc. A modelo realiza ações como, ativação dos processos de negocio ou mudar o estado da modelo. Após a ação do usuário e o resultado do processamento da modelo, o controlador escolhe uma visualização a ser exibida como parte da resposta a requisição do usuário, normalmente existe um controlador para cada funcionalidade relacionada.

Exemplo de interação entre as camadas



FONTE: <http://www.macoratti.net/vbn_mvc.htm>

A camada de apresentação exibe apenas a informação, não sendo de seu interesse onde ela foi obtida ou como a informação foi obtida.

* Elementos exibição no cliente: HTML , XML , ASP , Applets ;
* Camada de interface com o usuário;
* É utilizada na recepção de entrada de dados e exibe o resultado.

A Camada de lógica da Aplicação é o core do software, sendo sua responsabilidade tudo que a aplicação vai fazer.

* Modela comportamento por atrás do processo de negócios e os dados;
* É responsável apenas com o armazenamento, manipulando e gerando de dados;
* Encapsula os dados e seu comportamento independe da apresentação.

A Camada de Controle é responsável por controla o fluxo da apresentação, sendo uma camada intermediária entre a camada de apresentação e a lógica.

* Controla e mapeia as ações.

REFERÊNCIAS BIBLIOGRÁFICAS

<http://noticias.terra.com.br/brasil/politica/eleicoes/pesquisa-horario-politico-nao-influencia-60-dos-estudantes,0baedf0a2566b310VgnCLD200000bbcceb0aRCRD.html>

<http://www.jornaljovem.com.br/edicao4/tema14.php>.

<http://www1.folha.uol.com.br/poder/1159037-horario-eleitoral-nao-influencia-voto-dos-jovens-aponta-enquete.shtml>

<http://www.academia.edu/392273/Cientistas_Politicos_Comunicologos_e_o_Papel_da_Midia_nas_Teorias_da_Decisao_do_Voto>

BORBA, Felipe Moraes. (2005), *Razões Para a Escolha Eleitoral*

: Influência da Campanha Política na Decisão do Voto em Lula

FIGUEIREDO, Marcus Faria. (2007), “Intenção de Voto e Propaganda Política: Efeitos da Propaganda Eleitoral”,*Logos*, no. 27: 9-20.

<http://www.mountaingoatsoftware.com/system/asset/file/52/PortugueseScrum.pdf>

<http://www.caelum.com.br/apostila-java-web/>

<http://www.springframework.org>

MAGELA, R. **Engenharia de software aplicada**: fundamentos. Rio de Janeiro: Alta Books, 2006.

<http://www.caelum.com.br>

<http://www.eclipse.org>

<http://www.spring.io>

Vire o jogo com Spring Framework Autor, Henrique Lobo Weissmann, Casa do código

Scrum: Gestão ágil para projetos de sucesso, Rafael Sabbagh, Casa do código

Test-Driven Develpment + Design Patterns, Mauricio Aniche e Eduardo Guerra, Casa do código

Introdução à arquitetura e design de software: Uma visão sobre a plataforma Java, Paulo Silveira, Guilherme Silveira, Sérgio Lopes, Guilherme Moreira, Nico Steppat, Fabio Kung.

<http://acens.com.br/blog/linguagens-do-mercado-julho-de-2013/>

<http://www.devmedia.com.br/conceitos-fundamentais-de-banco-de-dados/1649>

<http://www.slideshare.net/fernandoabcampos/estudo-de-mapeamento-objetorelacional-com-framework-hibernate>

<http://www.devmedia.com.br/conheca-os-padroes-de-projeto/957>

[1] GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.

Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra. Head First Design Patterns. O'Reilly Media, 2004.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 2010.

 Alexander, Christopher. A Pattern Language. New York: Oxford University Press, 1977.

 JOSHUA KERIEVSKY. Refatoração para padrões. Bookman, ISBN: 9788577802449, 2008.

 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. 1 ed. Estados Unidos: Addison-Wesley, 1995. ISBN 0-201-63361-2.