POSSIBILIDADES E LIMITAÇÕES DA ARQUITETURA MVC (MODEL – VIEW – CONTROLLER) COM FERRAMENTA IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

SANTOS, Isaias[1], MOREIRA, Fábio Júnior[1], SILVA, João Antônio Albino[1], FREITAS, Marcos Tadeu[1]

REIS, José Cláudio de Sousa[2]

[1] Acadêmicos do 8º período do curso de Computação – UNIFENAS - Alfenas.

[2] Professor do curso de Computação, UNIFENAS - Alfenas.

Abstract. *The software architecture today is one of the critical issues of software construction and due to some shortcomings, there was the need to create new architectures. Among the new architectures comes the MVC architecture (Model-View-Controller) which appears increasingly present in the development environment for Web Due to its ease of creating unit tests, maintainability and more control of the generated code in an MVC architecture (Model-View-Controller) becomes a major architectures today. The goal is to demonstrate the use of the MVC architectural pattern, showing its advantages and disadvantages and the main differences between developing traditional layered. With the development of the project using the ASP.Net MVC Framework 2.0, allowed to demonstrate that the architecture provides a way to split the functionality involved in the maintenance and presentation of data from one application and the mapping of the traditional tasks of input, processing and output to the model interaction with the user. The adoption of the MVC. NET became more refined management of the code in Web applications.*

Key - words: MVC, software architecture, ASP.NET MVC Framework 2.0, Web

Resumo. *A arquitetura de software hoje em dia é um dos pontos críticos da construção de software e devido a algumas deficiências, surgiu a necessidade de se criar novas arquiteturas. Dentre as novas arquiteturas surge a arquitetura MVC (Model-View-Controller) que se mostra cada vez mais presente no ambiente de desenvolvimento para Web. Devido á sua facilidade de criar testes unitários, facilidade de manutenção e maior controle do código gerado a arquitetura MVC (Model-View-Controller) se torna uma das principais arquiteturas de hoje em dia. O objetivo é demonstrar a utilização do padrão arquitetural MVC, mostrando suas vantagens e desvantagens e as principais diferenças entre o desenvolvimento em camadas tradicional. Com o desenvolvimento do projeto utilizando o Framework ASP.Net MVC 2.0, possibilitou demonstrar que a arquitetura fornece uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação e o mapeamento das tarefas tradicionais de entrada, processamento e saída para o modelo de interação com o usuário. A adoção do MVC na plataforma .NET tornou mais apurado o gerenciamento do código nas aplicações Web.*

Palavras – Chave: MVC, Arquitetura software, Framework ASP.NET MVC 2.0, Web.

1. INTRODUÇÃO

Um dos pontos críticos na construção de software atualmente é a sua arquitetura que deve permitir a um sistema satisfazer as principais exigências: desempenho, confiabilidade, portabilidade e escalabilidade.

No passado , os softwares eram desenvolvidos para rodar apenas em uma única máquina, possuindo apenas uma camada em seu desenvolvimento, este tipo de arquitetura era conhecida como monolítico. Os principais problemas desta arquitetura são desenvolvimento e manutenção, pois a lógica do negócio e lógica de acesso a dados e os eventos de usuários ficam todas numa única camada.

Devido a este problema surgiu a necessidade de se criar outra arquitetura, surgindo então a arquitetura de duas camadas. A camada de dados ficaria numa camada específica e a lógica de negócio e interface do usuário em outra. Com o surgimento da internet, este padrão teve de ser alterado devido ao tempo necessário para carregar todos os componentes da regra de negócios no cliente em um aplicativo web, desta forma surge, então, a arquitetura em três camadas, com o objetivo de separar a lógica de negócios, a lógica de acesso a dados e a interface com o usuário, possibilitando com isso que os usuários acessem as aplicações sem ser necessário a instalação em suas máquinas, sendo necessário apenas um browser.

Para dar suporte a estas necessidades surgiu um padrão arquitetural denominado MVC (MODEL, VIEW, CONTROLLER), que auxilia na tarefa de manter separados os tipos de responsabilidades procurando, assim, um baixo acoplamento e uma alta coesão tornando o sistema escalável.

2. REFERENCIAL TEÓRICO

A partir dos anos 40, com a evolução dos sistemas computadorizados, grande parte dos esforços e custos era concentrada no desenvolvimento do hardware. No início dos anos 50, a partir do domínio da tecnologia do hardware, as preocupações passaram a se voltar para o desenvolvimento dos sistemas operacionais, surgiram então as primeiras concretizações destes sistemas, como também as linguagens de programação (FORTRAN, COBOL) e compiladores.

O objetivo desta época foi de facilitar o uso do computador (não sendo necessário ao usuário o conhecimento do hardware), permitindo com isso uma maior concentração na resolução dos problemas computacionais (MAZZOLA, 2006).

No início dos anos 60, surgiram os sistemas operacionais com características de multiprogramação, com isso houve um aumento na eficiência e utilidade dos sistemas computacionais. Esse crescimento contribuiu para que os preços do hardware caíssem; Devido a esse aumento na eficiência, surgiu então a necessidade de se desenvolver

grandes sistemas de software em substituição aos pequenos aplicativos utilizados até então.

A partir desta necessidade, surgiu um problema nada trivial devido à falta de experiência e a não adequação dos métodos de desenvolvimento existentes para pequenos programas, o que foi caracterizado como “Crise do Software”, mas por outro lado, permitiu o nascimento do termo “Engenharia De Software” (MAZZOLA, 2006).

2.1 Engenharia de Software

A Engenharia de Software surgiu em meados dos anos 70 numa tentativa de contornar a crise do software e dar um tratamento de engenharia (mais sistemático e controlado) ao desenvolvimento de sistemas de software complexos. Um sistema de software complexo se caracteriza por um conjunto de componentes abstratos de software (estruturas de dados e algoritmos) encapsulados na forma de procedimentos, funções, módulos, objetos ou agentes e interconectados entre si, compondo a arquitetura do software, que deverão ser executados em sistemas computacionais.

Os fundamentos científicos para a engenharia de software envolvem o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de software, avaliando e garantido suas qualidades. Além disso, a engenharia de software deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento (CAVALCANTI, [200-]).

Uma definição de Engenharia de Software proposta por Fritz Bauer:

“Engenharia de Software é a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que seja confiável e que trabalhem eficientemente em máquinas reais.” (PRESSMAN, 2006, p.17).

O IEEE define como

> Engenharia de Software: (1) aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software; isto é, a aplicação da engenharia ao software. (2) o estudo de abordagens como as de (1) (PRESSMAN, 2006, p.17).

A engenharia de software é uma tecnologia em camadas. Qualquer abordagem de engenharia (inclusive a engenharia de software) deve se apoiar num compromisso organizacional com a qualidade (PRESSMAN, 2006, p.17).

FIGURA 1 – Engenharia de software em camadas.

Fonte: PRESSMAN (2006, p.17)

O alicerce da engenharia de software é a camada de processo. O processo de engenharia de software é o adesivo que mantém unidas as camadas de tecnologia e permite o desenvolvimento racional e oportuno de softwares de computador (PRESSMAN, 2006, p.17)

Os processos de software formam a base para o controle gerencial de projetos de software e estabelece o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas.

Os métodos de engenharia de software fornecem a técnica de “como fazer” para construir softwares. Eles abrangem amplo conjunto de tarefas que incluem

comunicação, análise de requisitos, modelagem de projeto, construção de programas, testes e manutenção.

As ferramentas de engenharia de software fornecem apoio automatizado ou semi-automatizado para o processo e para os métodos (PRESSMAN, 2006, p.17)

2.2 ARQUITETURA DE SOFTWARE

Um ponto decisivo na construção e no projeto de todo o sistema de software é sua arquitetura: isto é, sua organização bruta como uma coleção de componentes de interação.

A arquitetura de software é uma concepção de fácil compreensão e que a maioria dos engenheiros entende de modo intuitivo, especialmente quando se tem um pouco de experiência. No entanto, é difícil defini-lo com precisão. Em particular, é difícil desenhar uma linha bem definida entre o design e a arquitetura — a arquitetura é um aspecto do design que se concentra em alguns recursos específicos. ARQUITETURA... (2001).

Em *An Introduction to Software Architecture*, David Garlan e Mary Shaw sugerem que a arquitetura de software é um nível de design voltado para questões que vão além dos algoritmos e das estruturas de dados da computação. A projeção e a especificação da estrutura geral do sistema emergem como um novo tipo de problema. As questões estruturais incluem organização total e estrutura de controle global; protocolos de comunicação, sincronização e acesso a dados; atribuição de funcionalidade a elementos de design; distribuição física; composição de elementos de design; escalonamento e desempenho; e seleção entre as alternativas de design.

Há mais a arquiteturar do que apenas a estruturar. A arquitetura é o conceito de nível mais alto de um sistema em seu ambiente. Ele também abrange a adequação à integridade do sistema, às restrições econômicas, às preocupações estéticas e ao estilo. Ele não se limita a um enfoque interno, mas leva em consideração o sistema

como um todo em seu ambiente de usuário e de desenvolvimento, ou seja, um enfoque externo. (WORKING GROUP ON ARCHITECTURE, 1999).

2.3 ARQUITETURA MVC

Junto com o desenvolvimento, a evolução e a forma de se fazer os programas, novas abordagens tiveram que ser criadas para facilitar a programação e garantir que a manutenção dos softwares, depois de prontos, torna-se mais fácil.

A partir disso, surgiu o conceito de divisão de tarefas, de forma com que, cada camada tenha seu próprio escopo e definição e que haja comunicação entre estas camadas de maneira controlada e eficiente.

O padrão MVC ajuda a criar aplicativos que separam os diferentes aspectos da aplicação (entrada lógica, lógica de negócios e lógica de interface do usuário), fornecendo uma rigidez entre esses elementos. O padrão especifica onde cada tipo de lógica deve estar localizado no aplicativo. A lógica da interface do usuário pertence no modo de exibição (view). Lógica de entrada pertence o controlador (controller). Lógica comercial pertence no modelo (model). Esta separação ajuda a gerenciar a complexidade quando se cria um aplicativo, pois permite que se concentre em um aspecto da aplicação num determinado momento. Por exemplo, pode-se concentrar no modo de exibição sem depender da lógica de negócios. ASP.NET...([200-]).

Além de gerenciar a complexidade, o padrão MVC torna mais fácil testar aplicativos do que testar um aplicativo baseado em formulários da Web. Por exemplo, em um aplicativo baseado em formulários Web, uma única classe é usada para exibir a saída e para responder à entrada do usuário. Escrever testes automatizados para aplicativos baseados em formulários Web pode ser complexa, porque para testar uma página individual, você deve instanciar a classe de página, todos os seus controles filho e classes dependentes adicionais no aplicativo. Com tantas classes instanciadas para executar a página, pode ser difícil escrever testes que se concentram exclusivamente

em partes individuais do aplicativo. Testes para aplicativos baseados em formulários Web, podem ser mais difíceis de implementar que testes em um aplicativo MVC. ASP.NET...([200-]).

A rigidez entre os três componentes principais (Model - View - Controller) de um aplicativo MVC também promove o desenvolvimento paralelo. Por exemplo, um desenvolvedor pode trabalhar no modo de exibição, um segundo desenvolvedor pode trabalhar sobre a lógica do controlador e um terceiro desenvolvedor possa se concentrar na lógica de negócios no modelo.

A arquitetura que está representada abaixo é uma implementação do modelo MVC. O modelo MVC está preocupado em separar a informação de sua apresentação. ASP.NET...([200-]).

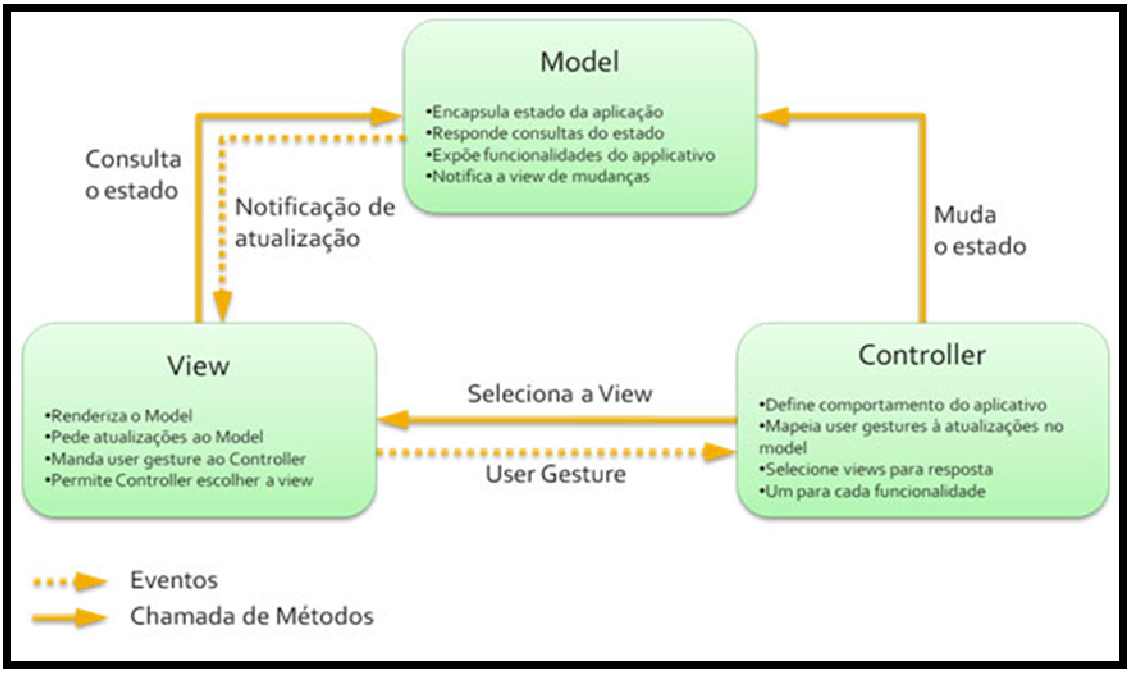

FIGURA 2 – Camadas MVC.

Fonte: ZEMEL (2009).

- **Model.** O *Model* é quem tem o contato com as informações armazenadas e que são mostradas, estejam elas armazenadas em um arquivo XML, banco de dados, ou onde quer que seja. É no *Model* e somente no *Model* que as

operações de CRUD (*Create, Retrieve, Update e Delete*, operações básicas usadas em bancos de dados relacionais) devem acontecer.

- **View.** É a parte de apresentação. É nesta camada que as informações, sejam elas quais forem e de qual lugar tenham vindo, serão exibidas para o usuário, acompanhadas de um bom *design*, uma boa estrutura organizacional, um ambiente agradável para quem está vendo, e muitos outros.
- **Controller.** É responsável por controlar todo o fluxo do programa. Fazendo uma analogia com o ser humano, o *controller* é como se fosse cérebro e o coração do aplicativo, é nele que se decide "se, o que, quando, onde" e tudo o mais que faz com que a lógica funcione. Sua função vai desde o que deve ser consultado no banco de dados até a tela que vai ser exibida para o usuário. (ZEMEL, 2009).

2.3.1 Como Ocorre a Comunicação Entre as Camadas MVC

- O usuário interage com a *View*
  - Quando o usuário age sobre a *View* (clicando em um botão, por exemplo), esta avisa ao *Controller* sobre a ocorrência dessa ação.
  - O *Controller* decide então o que fazer.
- O *Controller* pede ao *Model* para mudar seu estado
  - Quando o usuário clica um botão, o *Controller* interpreta o significado dessa ação e decide como o *Model* deve ser manipulado baseado nessa mesma ação. (VIDAL, 2008)
- O *Controller* também pode pedir *View* para mudar
  - O *Controller* poderia, por exemplo, habilitar ou desabilitar alguns botões ou itens de menu da interface de usuário.
- O *Model* notifica o *View* quando sofre mudanças no seu estado
  - Quando o estado do *Model* é alterado, como resultado de alguma ação do usuário ou de alguma ação interna da aplicação (por exemplo, fim da transmissão de um arquivo em um aplicativo de download), o *Model* notifica o *View* sobre essa mudança. (VIDAL, 2008)

- O *View* consulta o *Model* sobre seu novo estado
    - As informações exibidas pelo *View* são buscadas diretamente do *Model*.
    - O *View* pode buscar informações do *Model* ao ser avisado por ele de uma mudança de estado, ou como resultado de uma requisição do *Controller* sobre o *View*. (VIDAL, 2008)

Veja o exemplo:

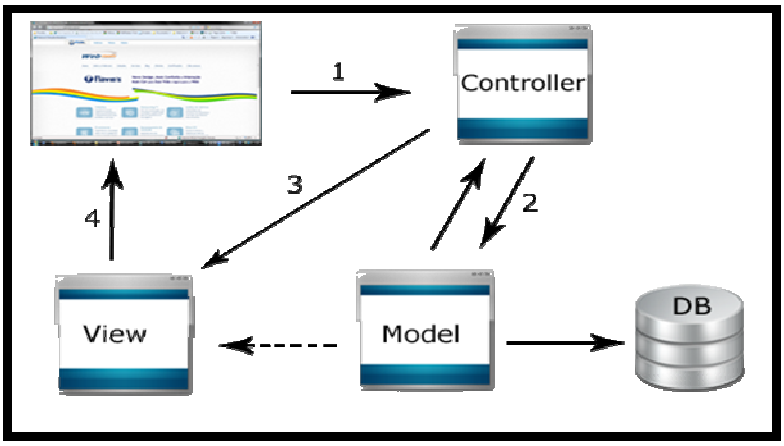

FIGURA 3 – Comunicação entre as camadas MVC.
Fonte: LEMOS (2009).

1º - O usuário faz uma requisição a uma página web (*View*) ao *Controller*. Exemplo: Preenchimento de um formulário de cadastro.

2 º - O *Controller* captura todas as informações da página web (*View*) e solicita uma ação para o *Model* (inserção, por exemplo). O *Model* processa a ação e devolve o resultado do processamento para o *Controller*.

3 º - O *Controller* processa o resultado e devolve para *View* o que ela deve exibir.

4º - A *View* recebe o que foi processado pelo *Controller* e exibe o resultado para o usuário. (LEMOS, 2009)

2.3.2 WebForm x MVC

Aplicação Web baseada em MVC

- Facilita gerenciar a complexidade, dividindo um aplicativo em modelo, modo de exibição e controlador.
- Ele não usa estado de exibição ou formulários baseados em servidor. Isso torna o framework MVC ideal para desenvolvedores que desejam total controle sobre o comportamento de um aplicativo.
- Ele usa um padrão de Front Controller que processa solicitações de aplicativo da Web através de um único controlador. Isso permite que você crie um aplicativo que ofereça suporte a uma infraestrutura de roteamento rica.
- Ele oferece melhor suporte para desenvolvimento controlado por testes (TDD).

Ele funciona bem para aplicativos da Web que são suportados por grandes equipes de desenvolvedores e Web designers que precisam de um alto grau de controle sobre o comportamento do aplicativo. ASP.NET...([200-]).

Aplicação Web Baseada em WebForm

- Ele suporta um modelo de evento que preserva o Estado em HTTP, que beneficia o desenvolvimento de aplicativos de Web de linha de negócios. O aplicativo baseado em WebForms oferece dezenas de eventos que são suportados em centenas de controles de servidor.
- Ele usa um padrão de Page Controller que adiciona funcionalidade a páginas individuais.
- Ele usa estado de exibição ou formulários baseados em servidor, que podem facilitar a gestão de informações de Estado.
- Ele funciona bem para pequenas equipes de desenvolvedores Web e designers que desejam aproveitar o grande número de componentes disponíveis para desenvolvimento rápido de aplicativos.

Em geral, é menos complexa para desenvolvimento de aplicativos, porque os componentes (a classe de página, controle, etc) são totalmente integrados e geralmente exigem menos código que o modelo MVC. ASP.NET...([200-]).

2.3.3 Vantagens e Desvantagens Envolvidas no MVC

Vantagens:

- Como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo é fácil manter, testar e atualizar sistemas múltiplos.
- É muito simples incluir novos clientes apenas incluindo seus visualizadores e controles.
- Torna a aplicação escalável.
- É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle, pois são independentes.

É mais fácil gerenciar a complexidade da aplicação dividindo-a em (MVC) modelo, visualizador e controlador (MACORATTI, 2004).

Desvantagens:

- Requer uma quantidade maior de tempo para analisar e modelar o sistema.
- Requer pessoal com conhecimento especializado.
- Não é aconselhável para pequenas aplicações. (MACORATTI, 2004)

3. MATERIAL E MÉTODOS

Etapa 1: Levantamento Bibliográfico.

Utilizando como método para a elaboração do projeto, pesquisas em sites, revistas e artigos relacionados ao tema.

Etapa 2: Escolha do sistema a ser construído

Definição do tema da aplicação a ser construída, utilizando o padrão arquitetural MVC.

Etapa 3: Desenvolvimento

Desenvolvimento da aplicação utilizando as ferramentas e tecnologias escolhidas.

Etapa 4: Análise

Análise do trabalho quanto às dificuldades encontradas e os resultados obtidos.

Etapa 5 : Conclusão

Conclusão do trabalho e considerações finais.

4. RESULTADO E DISCUSSÃO

Com a implementação do sistema, foi possível verificar que para um projeto de pequeno porte e de menor prazo de entrega, a arquitetura MVC não seria tão eficiente quanto ao uso das camadas tradicionais WebForm, pois ao se usar o padrão MVC demanda – se um tempo maior para analisar e criar o projeto .

Verificou – se que com a utilização do padrão arquitetural MVC tem - se um projeto mais complexo em comparação ao modelo tradicional, mas deve – se levar em conta que a longo prazo, o projeto pode se tornar mais facilmente escalável e com uma maior facilidade na manutenção e no reuso do seu código.

A arquitetura MVC oferece ao desenvolvedor os seguintes aspectos: uma clara separação entre as camadas da aplicação (divide as responsabilidades da equipe), reutilização de código e uma maior facilidade na manutenção do sistema.

5. CONCLUSÃO

Chegou - se a conclusão que a arquitetura MVC demonstrou eficiência na construção de aplicações, já que ela promove melhoria no processo de construção de software.

As melhorias que a arquitetura MVC trás em relação a arquitetura tradicional são refletidas na contrução do software, como a arquitetura realmente cumpre a clara separação das camadas, é permitido com isso o trabalho em paralelo, onde pode-se ter uma equipe trabalhando na parte do design visual (View), uma outra equipe trabalhando na parte de codificação (Controller) e outra equipe trabalhando na lógica de negócio (Model).

A arquitetura MVC propicia a construção de um código mais compacto e limpo, como não trabalha com componentes prontos, todo código a ser criado será apenas o que realmente vai ser utilizado, diferentemente da arquitetura tradicional WebForms, que ao se arrastar um componente, ele geralmente trás consigo algum código que não será utilizado.

Concluiu - se que com o desenvolvimento de software utilizando a arquitetura MVC, pode trazer no início um certo grau de complexidade, porém no final do desenvolvimento e posteriormente em sua manutenção o trabalho fica simples.

REFERÊNCIAS

ASP.NET MVC. [200-]. **Visão geral do ASP.NET MVC**. Disponível em: <http://www.asp.net/mvc/tutorials/asp-net-mvc-overview--cs >. Acesso em: 05 abril. 2010.

Arquitetura de Software. Disponível em: <http://www.wthreex.com/rup/portugues/process/workflow/ana_desi/co_swarch.htm> Acesso em: 25 abril. 2010.

BUSCHMANN, Frank; MEUNIER, Régin; ROHNERT, Han; SOMMERLAD, Peter; STAI, Michael. **Pattern-Oriented Software Architecture - A System of Patterns**. Nova York: John Wiley and Sons, 457 p.1996. ISBN :0-471-95869-7.

CAVALCATI, Jair. **Engenharia de Software**.[200-]. Disponível em: <http://www.dimap.ufrn.br/~jair/ES/index.html>. Acesso em: 20 março de 2010.

GAMMA, Erich et al. **Padrões de Projeto: Soluções reutilizáveis de software Orientado a Objetos**. Porto Alegre: Bookman, 2000.

LEMOS, Tiago. **O que é o MVC - Model View Controller**. 2009. Disponível em:

< http://www.tiagolemos.com.br/2009/7/10/o-que-e-o-mvc-model-view-controller/>. Acesso em: 20 abril de 2010.

MAZZOLA, Vitório. **Engenharia de software :Conceitos básicos**. Florianópolis: Editora da UFSC,c.1 140 p. 2006.

MACORATTI, José Carlos . **O modelo MVC – Model View Controller**. 2004. Disponível em: <http://www.macoratti.net/vbn_mvc.htm>>. Acesso em: 13 março de 2010.

ONEDA, Ricardo. 2007. **Considerações iniciais sobre o ASP.NET MVC Framework**.Disponível em: <http:/www.linhadecodigo.com.br/Artigo.aspx?id=1602> Acesso em: 29 fevereiro de 2010

PERRY, Dewayne. WOLF, Alexander. **Foundations for the Study of Software Architecture.** New York: ACM SIGSOFT Software Engineering Notes, V.17. 52 p. 1992.

PRESSMAN, Roger. S. ***Engenharia de Software***. São Paulo: Pearson Education do Brasil, 1995.

PRESSMAN, Roger. S. **Engenharia de Software.** 6. ed. São Paulo: McGraw-Hill, 2006.

VIDAL, Alexandre. **MVC - Model View Controller Architectural Pattern**. 2008. Disponível em: < http://www.deinf.ufma.br/~vidal/mvc.pdf>. Acesso em: 13 março de 2010.

WORKING GROUP ON ARCHITECTURE. 1999. **Recommended Practice for Architectural Description**. Disponível em:< http://www.iso-architecture.org/ieee-1471/introducing-p1471.pdf >. Acesso em: 5 Abril de 2010

ZEMEL, Tárcio. **MVC (Model – View – Controller).** 2009. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/>. Acesso em: 25 março. 2010.