APLICABILIDADE DA ARQUITETURA MVC EM UMA APLICAÇÃO WEB(WebApps)

LEMOS, Maxmilian Ferreira de[1], OLIVEIRA, Patrícia Carvalho de[1], RUELA, Leandro César[1], SANTOS, Matheus da Silva[1], SILVEIRA, Thallis Carvalho[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.

Resumo

Desde o nascimento da engenharia de software, as empresas de desenvolvimento estão buscando cada vez mais desenvolver software*s* de qualidade, dentro do prazo e do orçamento. Para suprir as deficiências encontradas no desenvolvimento, foram criadas novas arquiteturas. O uso de padrões de arquitetura vem sendo fortemente utilizado pelas empresas. Dentre eles, o *pattern* MVC (*Model-View-Controller*) é o que mais se destaca. Com o objetivo de explorá-lo, procurou-se abordar as vantagens e as desvantagens, a facilidade no desenvolvimento, a alta manutenabilidade, a testabilidade e o alto reaproveitamento de código. Para alcançar tais objetivos, buscou-se fazer um levantamento bibliográfico em sites e em livros visando à busca de informações para compreender melhor o *pattern* MVC. Foi desenvolvida uma *WebApplication* utilizando o Framewok Asp.Net MVC 4.0, possibilitando demonstrar as facilidades na manutenção e no teste de sistema. No processo de modelagem, o *pattern* MVC requer maior tempo para analisar e para modelar o sistema, o que não é aconselhável para pequenos projetos, porém essencial para grandes projetos cujo controle e cuja organização do desenvolvimento são obtidos de forma eficiente e satisfatória.

Palavras – Chave: *Pattern* MVC. Arquitetura de software. Framework ASP.NET MVC 4.0.

Abstract

Since the birth of software engineering, development companies are increasingly seeking to develop quality software, on time and on budget. To overcome the defects found in development, new architectures were created. The use of architectural patterns is being heavily used by businesses, including the pattern MVC (Model-View-Controller) is the one that stands out. In order to explore it, is intended to address the advantages and disadvantages, ease in development, high maintainability, testability and high code reuse. To achieve these goals, we sought to review the literature on websites and books aimed at seeking information to better understand the MVC pattern. One WebApplication was developed using Asp.Net MVC 4.0 framewok, allowing demonstrate the facilities maintenance and system testing. In the modeling process the MVC pattern requires more time to analyze and model the system, which is not advisable for small projects, but essential for large projects whose control and development organization is achieved efficiently and satisfactorily.
Keywords: Pattern MVC. Software Architecture. ASP.NET MVC Framework 4.0.

1 INTRODUÇÃO

Pode-se dizer que grandes sistemas de software não são concluídos, evoluem com o passar do tempo. Essas mudanças são necessárias para que o software mantenha sua utilidade, sua qualidade e a satisfação dos usuários.

Na década de 70, Alexander (1977, *apud* MACORATTI, 2002a) formulou o conceito de padrões de projeto. Projetistas de *software* passaram a aplicar a ideia de padrões de projeto só na década de 90, quando descobriram esse conceito que ele criou (1977, *apud* SOUZA, 2013). Padrão de projeto é uma forma prevista, estruturada e documentada da solução de um dado problema que é bastante comum, mesmo que em projetos e em áreas distintas.

Um dos pontos críticos da construção de software é a arquitetura de software. Antigamente, os softwares possuíam uma camada de desenvolvimento e eram projetados para rodar em uma máquina apenas. Essa arquitetura, chamada

monolítica, apresentava problemas de desenvolvimento, de manutenção e de teste, pois toda sua lógica, seus dados e eventos de usuários ficavam numa única camada.

Com o surgimento de novas tecnologias, tais como a Internet e as aplicações web (*World Wide Web*), novas arquiteturas fizeram-se necessárias. Entre estas novas arquiteturas, têm-se a arquitetura em camadas e a arquitetura MVC (*Model, View* e *Controller*).

2 REFERENCIAL TEÓRICO

2.1 Início do desenvolvimento de *software*

Nos primeiros anos da Computação, o software era projetado sob medida para cada aplicação e desenvolvido pelo próprio programador ou organização. O programador que desenvolvia era o mesmo que colocava em funcionamento e consertava os erros quando ocorriam (PRESSMAN, 1995).

O software passou a ter uma ampla distribuição na segunda era da evolução (de 1960 até o final da década de 1970), quando centenas ou milhares de programas foram distribuídos para usuários. Com esse crescimento, houve um número maior de exigência para alteração e ou adaptação quando um novo hardware era comprado. (PRESSMAN, 1995).

Com o avanço da tecnologia do hardware, softwares complexos e de grande porte passaram a serem desenvolvidos, entretanto vários projetos importantes que sofriam enormes atrasos, depois de implantados, eram difíceis mantê-los. Muitos não eram confiáveis, seus custos superavam as previsões e detinham um desempenho insatisfatório (SOMMERVILLE, 2007).

Iniciou-se, então, a crise do software que era consequência da indisciplina que dominava o mundo do desenvolvimento do software. Necessitava-se de que esse mundo fosse disciplinado, obtendo-se então a conclusão de que o processo de software precisava passar pelos processos da engenharia (PRESSMAN, 1995, 2011).

Em 1968, a OTAN (Organização do Tratado do Atlântico Norte) realizou uma conferência sobre Engenharia de Software, a NATO (North Atlantic Treaty Organization) Software Engineering Conference, em Garmisch, conforme ilustrado na FIG.1, na Alemanha, com o principal objetivo de estabelecer melhores práticas para o desenvolvimento. Esse encontro é considerado como o nascimento da Engenharia de Software (LUIZ, 2010).

FIGURA 1 - NATO *Software Engineering Conference* 1968

A engenharia de software é uma tecnologia em camadas, cujo foco na qualidade é o que sustenta a engenharia de software, sendo assim, a principal camada. Tudo o que estiver ligado ou relacionado à engenharia deve ser fundamentado na qualidade.

2.2 Processo de desenvolvimento

A construção de um software de alta qualidade depende dos passos que foram utilizados para se obter o seu resultado final. Esse roteiro que o software necessita seguir, de acordo com suas necessidades, é chamado de Processo de Software (PRESSMAN, 2006).

O esqueleto de um processo de software é chamado de arcabouço de processo. Ele contém cinco atividades de arcabouço ou atividades metodológicas, que podem ser aplicadas a todos os tipos de projetos de software:

- Comunicação: levantamento de requisitos e de atividades relacionadas, por meio da comunicação com o cliente e os demais interessados;
- Planejamento: estabelecimento do plano para o trabalho, no qual são descritos os prováveis riscos, as tarefas que devem ser seguidas, os produtos a serem produzidos e o cronograma do trabalho;
- Modelagem: um modelo é criado para auxiliar o desenvolvedor e o cliente a entender melhor os requisitos e o projeto de software;
- Construção: os códigos são gerados juntamente com os testes necessários para revelar possíveis erros;
- Entrega: entrega do software ao cliente para sua avaliação.

Para dar suporte ao desenvolvimento de software, percebeu-se que, além do processo, era necessário usar uma arquitetura.

"A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles" (PRESSMAN, 2011, p.230).

Segundo PRESSMAN (2011), existem vários estilos de arquitetura. Algumas delas são: arquitetura centralizada em dados; arquitetura de fluxo de dados; arquitetura de chamada e retorno; arquitetura orientada a objetos; arquitetura em camadas e arquitetura MVC.

2.3 Início do desenvolvimento em camadas

O padrão de desenvolvimento de software em camadas passou a ser utilizado como a principal ferramenta de arquitetura de sistemas, desde o surgimento da arquitetura cliente/servidor. Com o desabrochar da grande rede mundial (Internet), as precauções com a arquitetura aplicada passaram a ganhar uma grande importância (CHIBA; NARDI, 2007).

No desenvolvimento de sistema, componentes com responsabilidades iguais ou similares são agrupados na mesma camada (SILVA, 2011). Em décadas passadas, os eventos dos usuários, a lógica de negócios e os acessos a dados estavam presentes em uma única camada, dificultando, assim, o desenvolvimento e manutenção do software, e, por essa característica, foi denominado aplicação monolítica (BAPTISTELLA, 2009).

A aplicação em duas camadas surgiu com a necessidade de se compartilhar a lógica de acesso a dados entre vários usuários simultaneamente. A camada de acesso à base de dados se encontra em uma máquina especifica e a lógica de negócio com a interface com o usuário ficava em outra camada. Um grande problema era que aplicativos dos usuários precisavam ser atualizados quando alterações eram feitas (MACORATTI, 2002b).

Para solucionar o problema da arquitetura em duas camadas, a lógica de negócio é colocada em outra camada. Obtém-se, assim, uma camada para acesso à base de dados, uma para lógica de negócio e, por fim, outra para a interface com o usuário. Esse tipo é denominado aplicação em três camadas (BATTISTI, 2003).

2.4 Arquitetura *Model View Controller*

O padrão arquitetural MVC foi criado e introduzido pela primeira vez na Smalltalk-76 (Linguagem de programação) em 1978, por Reenskaug enquanto ele era um cientista visitante no Palo Alto Research Laboratory Xerox (PARC), como uma solução para o problema geral de dar aos usuários controle sobre suas informações. A documentação definida naquela época era de quatro divisões, *Model*, *View*, *Controller* e *Editor*. O *Editor* é um componente temporário que a *View* cria em demanda, como a interface, que se encontra entre a visão e os dispositivos de entrada, como mouse e teclado.

Na década de 1980, Jim Althoff e outros implementaram uma versão do MVC na Smalltalk-80, usando o *Controller* diferentemente de Reenskaug, observado pelo mesmo. Em implementações mais tarde feitas por Reenskaug, descrevendo a *View*, que aceita e lida com a entrada do usuário relevante para si, e o *Controller* aceita e lida com a entrada relevante para a montagem do *Controller/View* como um todo, assim passa a se chamar de Ferramenta a junção desses dois componentes.

A principal característica do MVC é facilitar para os usuários o emprego de uma aplicação, pois existe um vácuo entre o modelo mental e o modelo digital que existe no computador conforme ilustrado na FIG. 2. O MVC traz facilmente para o usuário a visualização e a manipulação dessas informações evidentemente. Assim, o usuário poderá ver o mesmo elemento do modelo em diferentes situações (REENSKAUG, 1978).

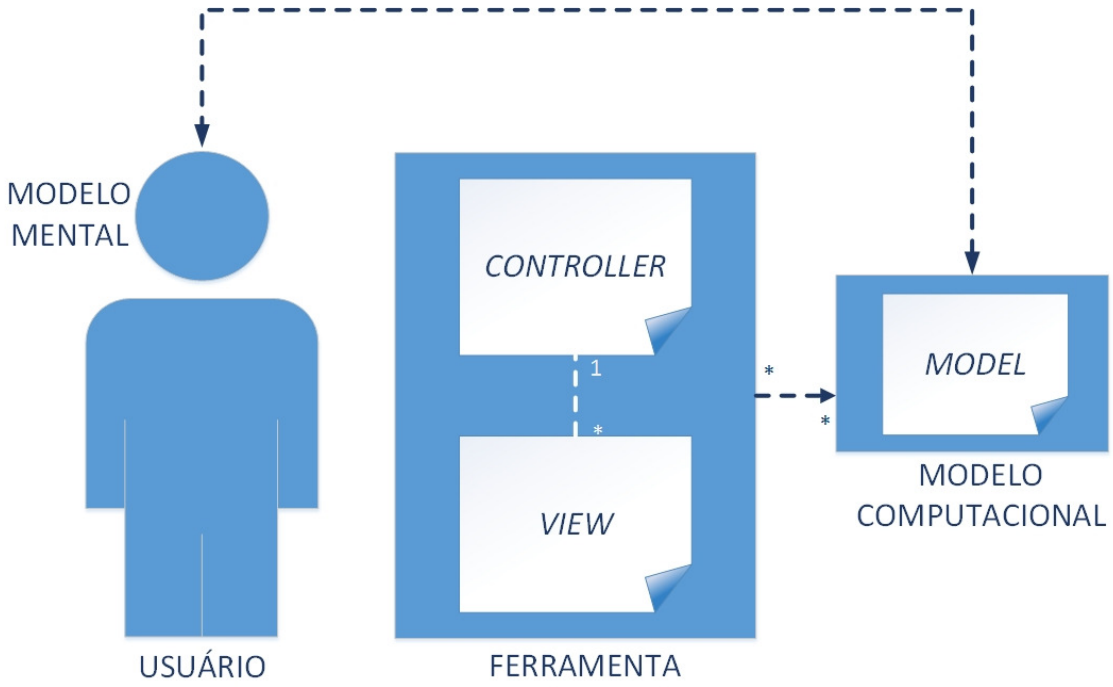

FIGURA 2 – Modelo mental MVC.

2.4.1 Características dos componentes *Model*, *View* e *Controller*

A arquitetura MVC auxilia os desenvolvedores a construir aplicações separando seus principais componentes, a manipulação e armazenamento dos dados, as funções que irão trabalhar com as entradas dos dados e a visualização do usuário. A arquitetura MVC especifica onde cada tipo de lógica deve estar localizada na aplicação (SANTOS et al., 2010).

O componente *Model* é o objeto de aplicação, a *View* é a interface visualizada pelo usuário e o *Controller* trabalha em relação às entradas de uma *View* e como as mesmas reagirão (GAMMA et al, 2000).

A *Model* contém a comunicação com os dados armazenados que serão visualizados na *View*, podendo estar armazenado em um banco de dados, em um arquivo XML ou em qualquer meio para tal funcionalidade. É somente na *Model* que as operações de *create*, *reader*, *update* e *delete* (CRUD), operações básicas em um banco de dados, podem ocorrer.

A *View* é a camada de apresentação da aplicação, o que será visualizado pelo usuário final, não importando quais dados e de qual lugar tenham vindo, mas, sim, de como serão exibidas essas informações.

A *Controller* é responsável por administrar todo o fluxo da aplicação, é o que move a aplicação, a lógica trabalha com os dados de entrada da *View* e resolve qual operação utilizará da camada *Model.* A FIG. 3 representa algumas funções e a comunicação de cada componente (ZEMEL, 2009).

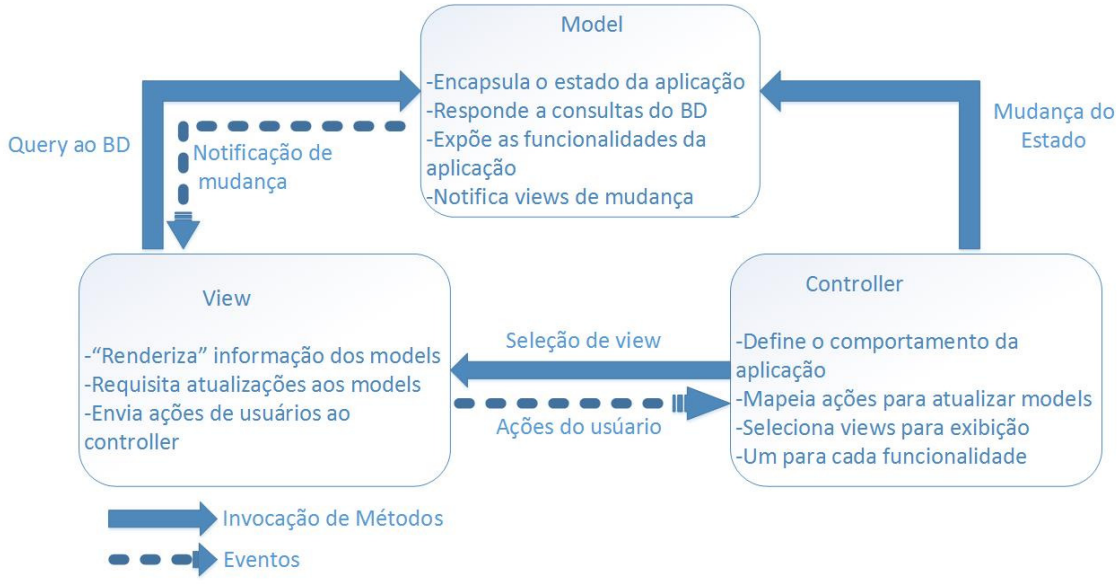

FIGURA 3 – Modelo MVC.

A arquitetura MVC está relacionada com a arquitetura da aplicação, e como os componentes *Model, View e Controller* se comunicam (MACORATTI, 2002b).

2.5 Diferenças entre a arquitetura em camadas e arquitetura MVC

Uma das regras fundamentais da arquitetura em camadas é que a camada de apresentação nunca se comunica diretamente com a camada de dados. Assim, toda a comunicação em um modelo de três camadas percorre a camada intermediária, propriamente dita. A comunicação entre as camadas na arquitetura em três camadas é conceituada como linear e bidirecional, por não dar voltas e por transportar a informação por um único caminho, havendo ida e volta. Diferentemente, na arquitetura MVC, conforme ilustrado na FIG.4, a comunicação entre as camadas ocorre de forma triangular: a *View* envia uma modificação para o *Controller;* após receber esse sinal, o *Controller* atualiza o *Model*, sendo a *View* atualizada diretamente do *Model.* Assim, a arquitetura MVC não é arquitetura em três camadas (MACORATTI, 2002b).

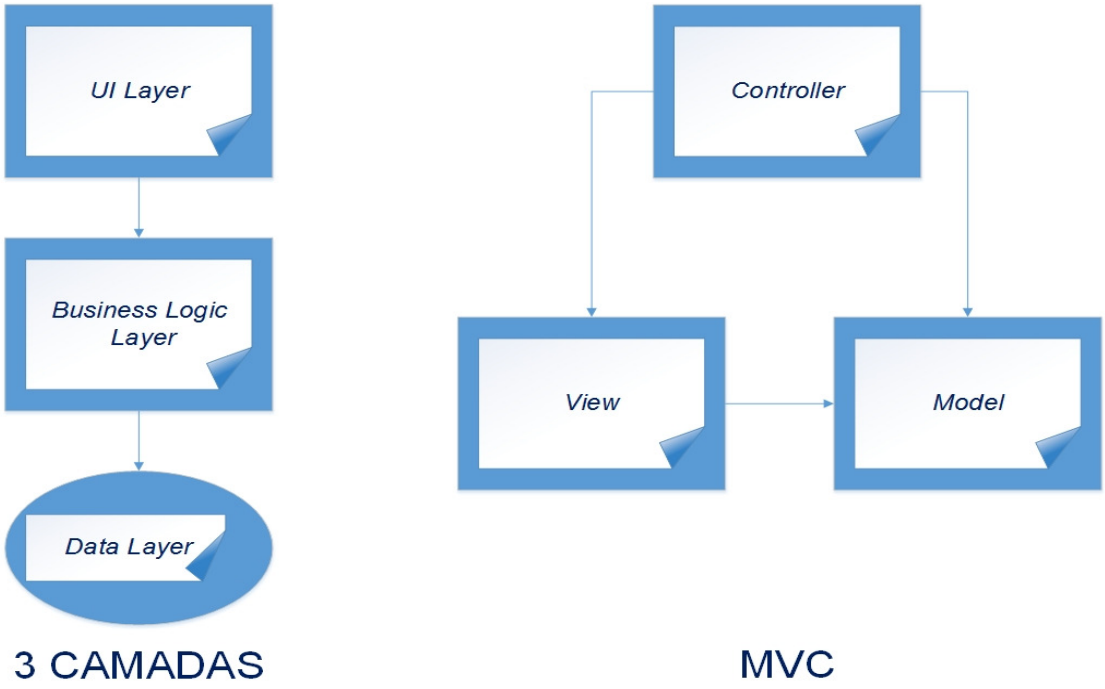

FIGURA 4 – Comunicação entre as camadas: MVC e 3 Camadas.

A complexidade da arquitetura MVC requer uma quantidade maior de tempo para analisar e para modelar o sistema, com uma equipe especializada. Não é aconselhável, portanto, para pequenas aplicações, pois a alta complexidade da arquitetura não irá trazer benefícios para tais aplicações (MACORATTI, 2002b).

Com a evolução dos computadores, da *Web* e de sistemas dinâmicos, foram implementados nos Framework*s* conceitos sobre várias arquiteturas e uma delas foi o MVC (RIBEIRO, 2013).

3 MATERIAL E MÉTODOS

Para dar início à realização do projeto, foi feita a escolha do tema a ser abordado. Definiu-se estudar o padrão arquitetural MVC e aplicá-lo no desenvolvimento de uma *WebApplication*.

Com o tema já definido, buscou-se fazer um levantamento bibliográfico em sites e em livros, visando à busca de informações para compreender e para enriquecer o desenvolvimento do trabalho.

Utilizando ferramentas e tecnologias escolhidas pelo grupo, foi possível dar início ao desenvolvimento da aplicação, e, então, fazer uma análise detalhada do assunto proposto, das dificuldades encontradas e dos resultados obtidos.

Com o desenvolvimento e a análise concluídos, passa-se para a produção da conclusão do trabalho, levando-se em conta se todos os seus objetivos foram cumpridos.

4 RESULTADO E DISCUSSÕES

4.1 As diferenças entre as arquiteturas MVC e três camadas

A arquitetura em 3 camadas divide suas tarefas em suas respectivas camadas: Camada de apresentação (UI), camada de lógica de negócio (BLL) e camada de persistência (DAL), possibilitando maior distribuição do processamento do sistema conforme mostra a FIG. 5.

Figura 5 – Três camadas.

A arquitetura em três camadas demonstra como agrupar os componentes por suas responsabilidades, sendo que a comunicação entre as camadas é bidirecional e linear, ou seja, utiliza somente um único caminho para enviar e retornar os dados, sempre passando pela camada intermediária (BLL).

Diferentemente da arquitetura em três camadas que agrupa os componentes por suas responsabilidades, a arquitetura MVC diz como a os componentes se comunicam em relação à arquitetura da aplicação.

Nota-se que o fluxo de comunicação da arquitetura MVC é triangular e unidirecional, ou seja, o cliente interage com a *View* enviando os dados para a *Controller*, que atualiza a *Model* com os dados e a *View* é atualizada com os dados da *Model,* em uma única direção, conforme FIG. 6.

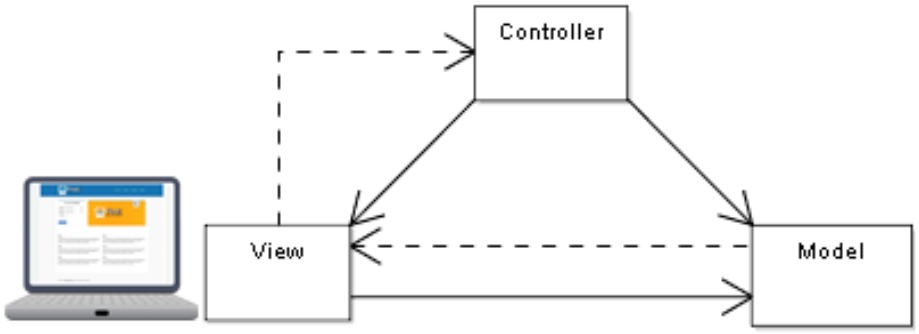

Figura 6 – Arquitetura MVC

4.2 Vantagens e Desvantagens

4.2.1 Arquitetura MVC

Vantagens:

O gerenciamento da complexidade se torna fácil devido à divisão da aplicação em *Model*, *View* e *Controller*. Esses componentes são independentes, sendo, então, possível o desenvolvimento paralelo.

A inclusão de novos clientes é realizada de forma simples, apenas incluindo seus visualizadores e controles, obtendo-se um melhor reaproveitamento do código. Não só a inclusão, mas também a customização e ou a substituição é facilitada, pois o MVC é completamente extensível.

Desenvolvedores que conhecem MVC terão facilidade com o código, pois o mesmo segue um padrão mais legível (legibilidade).

Desvantagens:

Desenvolvedores com pouca ou com nenhuma experiência em desenvolvimento de aplicações web encontram dificuldades no desenvolvimento, devido à ausência de controles prontos, ao modelo de programação orientada a eventos e *ViewState*. Assim, para projetos que irão utilizar o padrão MVC, a equipe terá de ter conhecimento especializado.

4.2.2 Arquitetura em três camadas

Vantagens:

Redução da complexidade e o favorecimento a coesão são as vantagens, pois os componentes são agrupados de acordo com suas responsabilidades e, entre eles, a comunicação é simplificada.

As camadas podem ser reutilizadas em outros sistemas ou substituídas, promovendo, assim, a reusabilidade. Caso a interface entre as camadas seja mantida, a implementação de uma camada pode ser mudada sem afetar outra camada, alcançando-se, assim, o princípio baixo acoplamento.

Desvantagens:

Número maior de classes existentes no sistema. Aplicações simples podem se tornar extremamente complexas, caso haja um exagero na criação de camadas. Alterar um nome de campo ou de tabela implicará diversas partes da sua aplicação.

5 CONCLUSÃO

Por intermédio da pesquisa realizada e do desenvolvimento de uma aplicação, pôde-se perceber que a arquitetura MVC possui mais vantagens do que desvantagens e que é recomendada não só para grandes aplicações, mas para qualquer sistema onde poderão ser adicionados novos recursos e realizadas diversas manutenções.

Na modelagem da aplicação, foi preciso dedicar mais tempo e atenção em consequência da utilização da arquitetura MVC, mas sendo possível subdividir as responsabilidades entre os integrantes em virtude de seu baixo acoplamento. A adição de outros padrões foi realizada sem problemas e, em alguns casos, foi possível o reaproveitamento do código com sucesso.

Conclui-se que, no desenvolvimento de um sistema utilizando a arquitetura MVC, tornou-se de fácil manutenibilidade, com adição de recursos, com reaproveitamento de código, com maior integração da equipe e com divisão das tarefas, usando-se sempre boas práticas de programação.

REFERÊNCIAS

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., ANGEL, S. **A Pattern Language**. New York, NY (USA): Oxford University Press, 1977.

BAPTISTELLA, Antônio José. **Abordando a arquitetura MVC, e Design Patterns: Observer, Composite, Strategy.** Set. 2009. Disponível em: <http://www.linhadecodigo.com.br/artigo/2367/abordando-a-arquitetura-mvc-e-design-patterns-observer-composite-strategy.aspx>. Acesso em: 29 agosto de 2013.

BATTIST, Júlio. **Criando aplicações em 3, 4 ou n Camadas**. Mai. 2003. Disponível em: < http://www.juliobattisti.com.br/artigos/ti/ncamadas.asp>. Acesso em: 30 agosto de 2013.

CODEIGNITER BRASIL. **Modelo MVC.** Disponível em: <http://codeigniterbrasil.com/wp-content/uploads/2008/10/mvc-model-view-controller-esquema-visual.gif> Acesso em: 7 Junho de 2013.

CHIBA, Cláudio; NARDI, Alexandre. **Desenvolvimento em camadas.** Mai. 2007. Disponível em: <http://www.microsoft.com/brasil/msdn/tecnologias/arquitetura/Layers_Developing.mspx>. Acesso em: 29 julho 2013.

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

IMASTERS. **NATO Software Engineering Conference 1968**. Disponível em: <http://conteudo.imasters.com.br/18732/nato%201.jpg> Acesso em: 7 Junho de 2010.

LUIZ, Ricardo. **Sem boas práticas de engenharia não há agilidade:** 2010. Disponível em: <https://www.ibm.com/developerworks/community/blogs/fd26864d-cb41-49cf-b719-d89c6b072893/entry/sem_boas_pr_C3_A1ticas_de_engenharia_n_C3_A3o_h_C3_A1_agilidade2?lang=pt_br>. Acesso em: 14 agosto de 2013.

MACORATTI, José Carlos. **Padrões de Projeto: Design Patterns.** 15 mai. 2002a. Disponível em: <http://www.macoratti.net/vb_pd1.htm>. Acesso em: 20 de maio de 2013.

_____. **Padrôes de projeto: O modelo MVC - Model View Controller**. 1 Jun. 2002b. Disponível em: <http://www.macoratti.net/vbn_mvc.htm>. Acesso em: 7 junho de 2013.

_____. **Comunicação entre as camadas: 3 Camadas e MVC.** Disponível em: Adaptado de < http://www.macoratti.net/vbn_mvc5.gif> Acesso em: 7 Junho de 2013.
NAUR, Peter; RANDALL, Brian. **Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee**. NATO, 1969

OSLO. **Modelo mental MVC**. Disponível em: Adaptado de <http://heim.ifi.uio.no/~trygver/themes/mvc/MVC-2006.gif > Acesso em: 7 Junho de 2013.

PRESSMAN, Roger. S. **Engenharia de software.** 3. ed. São Paulo: Makron Books, 1995.

_____. **Engenharia de software.** 6. ed. São Paulo: McGraw-Hill, 2006.

_____. **Engenharia de software:** Uma abordagem profissional. 7. ed. São Paulo: McGraw-Hill, 2011.

REENSKAUG, Trygve M. H. **MVC XEROX PARC**.1978. Disponível em: <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>. Acesso em: 7 Junho de 2013.

RIBEIRO, Rubens Takiguti. **MVC:** a essência e a web. 2013. Disponível em: <http://rubsphp.blogspot.com.br/2013/02/mvc-essencia-e-web.html>. Acesso em: 7 junho de 2013.

SANTOS, Isaias et al. **Possibilidades e limitações da arquitetura mvc (model – view – controller) com ferramenta ide (integrated development environment).** 2010. 56f. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) - Universidade José do Rosário Vellano, Alfenas, Mg.

SILVA, Thiago F. **MVC não é sobre camadas.** Jan. 2011. Disponível em: <http://tiagodev.wordpress.com/2011/01/29/mvc-nao-e-sobre-camadas/#comments>. Acesso em: 29 agosto 2013.

SOMMERVILLE, I. **Engenharia de software**. 8. ed. São Paulo: Addison-Wesley, 2007.

SOUZA, Cleidson. **Padrões de software (Software Patterns).** Disponível em: <http://www.ufpa.br/cdesouza/teaching/labes/padroes-de-software.pdf>. Acesso em: 30 maio de 2013.

ZEMEL, Tárcio. **MVC (Model – View – Controller).** 2009. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/>. Acesso em: 7 Junho de 2013.