FACULDADES INTEGRADAS DE CARATINGA

FACULDADE DE CIÊNCIA DA COMPUTAÇÃO

UTILIZAÇÃO DE METADADOS MYSQL NA GERAÇÃO DE CÓDIGOS COM PHP

LEONARDO WESLEI DINIZ

CARATINGA
2010

Leonardo Weslei Diniz

UTILIZAÇÃO DE METADADOS MYSQL NA GERAÇÃO DE CÓDIGOS COM PHP

Monografia apresentada ao Curso de Ciência da Computação das Faculdades Integradas de Caratinga como requisito parcial para obtenção do título de Bacharel em Ciência da Computação orientado pelo professor Glauber Luis da Silva Costa.

Caratinga
2010

Leonardo Weslei Diniz

UTILIZAÇÃO DE METADADOS MYSQL NA GERAÇÃO DE CÓDIGOS COM PHP

Monografia submetida à Comissão examinadora designada pelo Curso de Graduação em Ciência da Computação como requisito para obtenção do grau de Bacharel.

_____

Prof. Glauber Luis da Silva Costa

Faculdades Integradas de Caratinga

_____

Prof.

Faculdades Integradas de Caratinga

_____

Prof.

Faculdades Integradas de Caratinga

Caratinga, ___/___/___

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me guiado até aqui, a minha esposa Aline Ferraz Martins Diniz pelo apoio, a toda a minha minha família, aos que acreditaram em mim e aos meus pais, em especial minha mãe que deu o que ninguém nunca irá tirar de mim: o conhecimento.

O espaço é pequeno para colocar o nome das pessoas que tenho por terem contribuído de alguma forma para essa minha caminhada, mas a gratidão que tenho por elas não tem tamanho, obrigado.

Nossas dúvidas são traidoras e nos fazem perder o que, com frequência, poderíamos ganhar, por simples medo de arriscar.

William Shakespeare

RESUMO

O mundo está em contante transformação, a cada segundo as informações vão se multiplicando e cada vez mais há necessidade de tecnologia para organizá-las. Tudo está acontecendo muito rápido, tecnologias que até então eram consideradas novas rapidamente se tornam obsoletas.

Os desenvolvedores também sentem o efeito desta transformação, cada vez mais o mercado exige mais deles em um tempo cada vez menor. Quem não se adapta não sobrevive as transformações. Para cumprir prazos muitos usam *frameworks,* tem resultados rápidos, porém muitas vezes ficam presos à filosofia dos mesmos.

O presente trabalho propõe uma maneira de gerar códigos usando a linguagem de programação PHP a partir de metadados de um banco de dados MySQL, possibilitando ao desenvolvedor definir como cada linha de código será apresentada através de um *template* definido pelo mesmo. O intuito deste trabalho não é substituir *frameworks*, pelo contrário, poderá até mesmo caminhar apoiá-los na geração de códigos de acordo com seus padrões.

Este trabalho procura oferecer ao desenvolvedor um meio flexível de se trabalhar como códigos relacionados a bancos de dados, trazendo uma abstração geral de um banco de dados para os mesmo através do uso de um *template* que poderá ser replicado para todas as tabelas de acordo com as características de cada uma.

Palavras-chave: PHP, MySQL, banco de dados, *scaffolding*, gerador de código.

ABSTRACT

The world is in constant transformation. At every second the informations are growing and more technology is needed to organize it. Everything is happening very fast and new technologies has becoming obsolet rapidly.

The developers are also feeling the transformation effects, the market requires more productivity in less time ever. Who is not prepared to changes, doesn't survive to the transformations. To deal with due date, many developers are using frameworks, which gives him fast outcomes, but in many times, they get prisioners of its philosophy.

The current work proposes a way to generate codes using the PHP programming language through metadata of mysql database, giving to developers the possibility to define how each code line will be presented via a template defined by him. The goal of this work isn't be a framework substitution tool, but be a helpfull tool to generate codes according with its patterns.

This work proposes a flexible way to developers works on programming codes related to data bases, bringing him a general abstraction through the use of a template that could be replicated to all the tables of an data base according with the features of every table.

Keywords: PHP, MySQL, database, scaffolding,code generator.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

AJAX	*Asynchronous JavaScript And XML*
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
ASP	*Active Server Pages*
BD	Banco de dados
CASE	Computer-Aided Software Engineering
CGI	*Common Gateway Interface*
CRUD	*Create, Retrieve, Update and Delete*
DCL	*Data Control Language*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
HTML	*HyperText Markup Language*
ISO	*International Organization for Standardization*
JSP	*Java Server Pages*
LP	Linguagem de Programação
MVC	*Model-View-Controller*
OO	Orientado à Objetos ou Orientação à Objetos

PHP	*Hypertext Preprocessor*
RAD	*Rapid Application Development*
SEQUEL	*Structured English Query Language*
SGBD	Sistemas de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
URL	*Uniform Resource Locator*
XHTML	*eXtensible HyperText Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente o mercado de desenvolvimento de *software* enfrenta um grande desafio, o desenvolvimento de softwares bem estruturados e reusáveis de forma rápida, que cumpra seu objetivo com sucesso.

Em um mundo cada vez mais globalizado o tempo é um dos principais ingredientes para o sucesso de um projeto, porém, o tempo impacta diretamente nos custos de desenvolvimento de *softwares* e este fator é considerado no momento de se determinar como, e o que será produzido. Muitas vezes a falta de tempo resulta em *softwares* falhos que necessitam de constantes correções para que o seu funcionamento não seja comprometido.

Por outro lado o avanço da tecnologia trás também benefícios como por exemplo o surgimento de novas linguagens de programação, mais robustas e de altíssimo nível. Os Sistemas de Gerenciamento de Banco de Dados (SGBD), que fazem o armazenamento e tratamento dos dados usados pelos softwares e ferramentas CASE[1], que auxiliam os desenvolvedores em suas tarefas de projetos e desenvolvimento. Recursos como estes mostram aos desenvolvedores um novo caminho, que lhes possibilita desenvolver aplicações ágeis, robustas e com qualidade em curto prazos.

Uma das vantagens de se trabalhar com Banco de Dados (BD) é que o grau de abstração que se tem do problema, pois esta abstração torna o BD mais compreensível e manipulável, estes ainda podem ser modelados graficamente através de ferramentas CASE, aumentando ainda mais o grau de abstração. Como consequência, esta abstração trás a possibilidade de uso dos mesmos como documentação auxiliar para iniciar o desenvolvimento do *software*.

Mesmo com tecnologias avançadas e conhecimentos específicos, os desenvolvedores ainda encontram desafios, levando em consideração que rotinas

1 Do inglês *Computer-Aided Software Engineering* é uma classificação que abrange todas ferramentas baseada em computadores que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes.

empresariais nem sempre são estáveis, portanto o mercado exige cada vez mais que os softwares sejam flexíveis e de fácil uso.

O desenvolvimento de códigos pode ser exaustante, considerando que são necessários no mínimo quatro tipos diferentes de códigos-fonte para manipular os dados de uma tabela (baseado nas quatro operações básicas de um BD – inserir, recuperar, modificar e apagar). Partindo da suposição que a maioria das aplicações possuem varias tabelas relacionando entre si, é quase certo que é gasto muito tempo no desenvolvimento da base da aplicação. Portanto se não existir nenhuma ferramenta de auxilio para o desenvolvimento, dificilmente prazos serão cumpridos.

Outro grande problema é a portabilidade e as tecnologias usadas, pois, muitas empresas podem não ter um ambiente de adequado para a aplicação, muitas destas podem possuir suas próprias tecnologias ou algo que desejam que sejam aplicadas no desenvolvimento do novo software.

Já existem vários frameworks na área de desenvolvimento de software que fazem esta tarefa, mas cada qual muitas vezes com seus padrões e tecnologias e obrigam os desenvolvedores a trabalhar de acordo com eles.

A ideia principal deste estudo é fornecer uma ferramenta de auxilio ao desenvolvimento de software que possibilite gerar código utilizando os metadados de um banco de dados relacional e que possibilite ao desenvolvedor trabalhar da forma como achar melhor.

Espera-se encontrar uma forma de extrair informações do SGBD que possam ser aplicadas através de *templates* para gerar códigos.

2 REFERENCIAL TEÓRICO

As seções seguintes descrevem os aspectos teóricos citados no trabalho como: banco de dados, linguagens de programação, *frameworks*, *scaffolding*, geradores de código, qualidade de software e padrões de arquitetura de software.

2.1 BANCO DE DADOS

Atualmente os desenvolvedores contam com muitas tecnologias para auxiliá-los em seus projetos, entre elas os SGBD. O uso de SGBD trás uma visão diferenciada do projeto, ele separa a informação da programação, o que possibilita uma fácil manipulação de dados.

O conceito de SGBD pode ser descrito como a coleção de dados logicamente coerentes que possui um significado implícito e que cuja interpretação é dada por uma determinada aplicação, representa abstratamente uma parte do mundo real, que é de interesse de uma certa aplicação.

Um sistema de banco de dados tem por objetivo manter informações armazenadas para que seja disponibilizada quando solicitada, e ainda “garantir a segurança das informações armazenadas contra eventuais problemas com o sistema, além de impedir tentativas de acesso não autorizadas. Se os dados são compartilhados por diversos usuários, o sistema deve evitar a ocorrência de resultados anômalos” (SILBERSCHATZ et al.,1999).

> Podemos caracterizar o SGBD como um recurso de software composto por programas e utilitários destinados às tarefas voltadas para o completo

> gerenciamento de um sistema de banco de dados. As principais tarefas a serem desempenhadas pelo SGBD se constituem no armazenamento, organização, atualização e restauração de banco de dados de sistemas computacionais.(SILVA, 2001)

Em sistemas de computadores, quatro componentes formam um sistema de banco de dados, também conhecido como Sistema de gerenciamento de banco de dados (SGBD): dados, hardware, software e usuário.

2.1.1 Histórico

Segundo SILVA(2001, p.16), durante a década de 60, os paradigmas de armazenamento e processamento de informações passaram por uma grande mudança. Surgiram tecnologias de armazenamento que fizeram com que dados e aplicações de *software*, que antes formavam um único elemento, fossem separados, passando a trabalhar de forma independente um do outro. Isso permitiu a criação de ferramentas capazes de gerenciar e manipular dados de forma mais eficiente a fim de obter os melhor quantidade nos resultados.

Os sistemas de banco de dados surgiram na década de 60, oferecendo recursos de armazenamento e tratamento de informações de forma segura, rápida e eficiente. Deixando para trás as limitações da tecnologia baseada nos sistemas de arquivos e suas limitações.

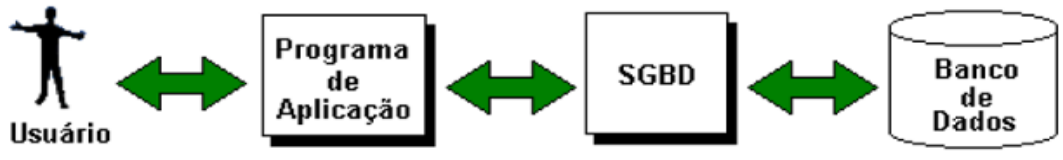

Figura 1: Representação de um sistema de banco de dados (SILVA, 2001)

Com os SGBDs houveram mudanças relativamente grandes como o acesso as informações. As aplicações que acessavam os dados diretamente e que passaram a ter acesso aos dados mediados pelos SGBDs através de requisições na forma de serviços.

2.2 LINGUAGEM DE CONSULTA ESTRUTURADA

A Linguagem de Consulta Estruturada (*Structured Query Language* - SQL), é a linguagem usada pelos bancos de dados relacionais. Essa linguagem surgiu nos anos 70, desenvolvida pela IBM e foi inicialmente chamada de SEQUEL, (*Structured English Query Language*, Linguagem de Consulta Estruturada em Inglês).

O SQL foi determinado como a linguagem padrão para bancos de dados relacionais, pela simplicidade e facilidade de uso e rapidamente foi adotada por outras empresas de bancos de dados.

Apesar de ter sido inicialmente elaborada pela IBM, veio a necessidade de padronização, devido a grande variedade de dialetos incorporados pelas demais empresas, essa tarefa foi realizada nos anos 86 e 87 pelo Instituto Nacional Americano de Padronização (*American National Standards Institute* - ANSI) e Organização Internacional para Padronização (*International Organization for Standardization* – ISO).

Em 1992, passou por uma revisão na linguagem, que passou a ser chamada de SQL-92, o mesmo se repetiu em 1999 e 2003, se chamando SQL-99 e SQL-2003. As maiores mudanças ocorridas nessas revisões, foram a utilização de expressões regulares e algumas características de orientação a objetos.

Muitos fabricantes de SGBDs optaram pelo padrão SQL, mas desenvolveram variações para uso em suas ferramentas. Em termos estruturais não há grandes variações, o que possibilita a migração de um padrão para outro, quando

necessário.

A SQL está dividida em operações de definição, manipulação e controle de dados. (DATE, 2003).

- Linguagem de definição de dados - a Linguagem de Definição de Dados (*Data Definition Language* - DDL) corresponde aos conjuntos de instruções que definem o banco de dados, como criação de tabelas, campos, tipos de dados, etc. Através das DDLs são também definidas as regras de integridade de dados.

- Linguagem de manipulação de dados - depois de definido o banco e sua estrutura através das DDLs, usa-se a Linguagem de Manipulação de Dados (*Data Manipulation Language* - DML), que faz operações relacionadas aos dados como: inserir, alterar, apagar e consultar dados. A DML é o subconjunto da SQL mais usado numa aplicação, pois é através delas que se faz requisições ao SGBD.

- Linguagem de controle de dados - a Linguagem de Controle de Dados (*Data Control Language* - DCL), é uma linguagem que estabelecendo regras de acesso e permissão aos dados.

Outra característica importante no padrão SQL, é que o mesmo dispõe de um conjunto de especificações para um catálogo de dados, conhecido como metadados, que “[...]consiste em um conjunto de tabelas SQL cujo conteúdo reproduz efetivamente, de uma forma definida com precisão, todas as definições de todos os outros esquemas no catálogo em questão.” (DATE, 2003).

2.3 MODELOS DE REPRESENTAÇÃO DE DADOS

O modelo de representação de dados está diretamente relacionado à

qualidade do SGBD, pois descreve como os dados serão acessados e manipulados pelos usuários dando a eles a capacidade de traduzir ou modelar o mundo real.

Atualmente, o modelo relacional é o mais usado, pois, além de solucionar os problemas encontrados nos modelos que o antecederam, fornecem mais flexibilidade na organização e manipulação de bancos de dados de maior porte e com mais complexabilidade.

Com o avanço da tecnologia e da demanda por sistemas capazes de gerenciar dados complexos foi sendo um grande incentivo para criação de novos modelos de tratamento de dados.

2.3.1 Bancos de dados relacionais

Este modelo é considerado o mais usado nas aplicações atuais, surgiu durante a década de 80 como um novo conceito para o banco de dados, onde se poderia representar entidades do mundo real como tabelas de um banco de dados.

Segundo ALMEIDA et al. (2007) o “modelo relacional é baseado nas teorias matemáticas, em que uma tabela é uma relação. No modelo relacional todo o acesso é abstraído através de instruções SQL, e o SGBD se responsabiliza por tratar essas instruções no nível mais baixo”.

2.3.2 Bancos de dados orientados a objeto

O modelo de BD orientado a objetos (OO) teve origem na combinação de ideias dos modelos de dados tradicionais e de linguagens de programação orientada a objetos, onde as tabelas do modelo relacional são consideradas objetos, e o acesso a esses objetos se estabelece através de classes.

De acordo com SILBERSCHATZ et al. (1999), a filosofia do modelo de dados OO consiste em agrupar os dados e o código que manipula estes dados em uma única entidade (objeto), estruturando-os de forma que possam ser agrupados em classes. Isso significa que, baseado nos conceitos OO, os objetos de banco de dados podem usar o mesmo mecanismo de herança usado para definir superclasses e subclasses de objetos, criando assim hierarquias.

Este modelo tem um papel importante nos SGBDs, pois são os mais adequados para o tratamento de dados complexos e dinâmicos e possuem maior naturalidade conceitual e, finalmente, por estarem em consonância com fortes tendências em linguagens de programação e engenharia de *software*.

2.3.3 Bancos de dados objeto relacionais

O modelo de BD objeto relacionais surgiu com o objetivo de expandir o modelo relacional para lidar com dados complexos não suportados pelo modelo relacional. De acordo com DATE (2003) “a ideia geral em cada caso é que o produto deve admitir recursos tantos de objetos quanto relacionais”.

> Como principal benefício, esse modelo permite ampliar a capacidade dos SGBDs para lidar com dados complexos, evitando que os produtores de software desconsiderem os investimentos feitos na produção de SGBDs relacionais. Entretanto, da mesma forma que o modelo OO, não existe um modelo padrão para construção de SGBDs objeto relacionais. Diferente dos SGBDs relacionais, que são sustentados por um modelo formal já definido, os SGBDs objeto relacionais. obtiveram sucesso comercial graças às

> iniciativas de implementações em produtos comerciais já disponíveis no mercado como o Oracle 10 da Oracle Corporation. (RIVELO, 2004).

2.3.4 Metadados

De acordo com metadados TAYLOR (1999) e DEMPSEY (1998) “são dados que descrevem completamente os dados que representam, permitindo ao usuário decidir sobre a utilização desses dados da melhor forma possível”.“A palavra metadados foi criada por Jack Myres em 1969, para denominar os dados que descreviam registros de arquivos convencionais” (HOWE, 1996).

Segundo ARELLANO (2010) as vantagens dos metadados são:

- Os metadados administram uma grande quantidade de dados
- Ajudam na descoberta, recuperação e edição efetiva dos recursos de informação na rede.
- Garantem a segurança de qualidade
- Compartilham e integram fontes de informação heterogêneas na transferência de aplicação para aplicação
- Podem ser traduzidos para uma mesma sintaxe
- Facilita a catalogação descritiva.

Metadados podem ser aplicados a vários acervos de dados. Seu objetivo principal é organizar e documentar dados minimizando esforços e facilitando a manutenção dos dados.

> Os metadados visam cumprir a função básica de prover informação sobre o documento digital, alimentando os processos de gestão, recuperação e reprodução.
> São fundamentais para a redução dos riscos e o aumento das chances de sobrevivência da informação digital.
> São fundamentais para o provimento da interoperabilidade necessária à

> explosão dos recursos de informação na *Internet*.
> São ferramentas que melhoram significativamente o trabalho na área dos dados e de grande importância para os usuários de Sistemas de Informação.(REIS, 2010).

2.3.4.1 Metadados na definição do banco de dados

Os SGBDs foram criados com a finalidade de armazenar os dados em tabelas de forma organizada e facilitar a realização de tarefas como: atualização, remoção e consulta a qualquer informação contida no mesmo.

O armazenamento de informações em tabelas, projetadas em conformidade com as restrições (regras) do modelo relacional (Date, 2000; Silberschatz et al., 2005), em especial com relação ao projeto normalizado do banco de dados, garante maior agilidade e flexibilidade para acesso as informações.

Com a utilização das funcionalidades oferecidas pelo SBGD MySQL (Axmark et al., 2001) tem-se a garantia de consistência, integridade e de segurança no acesso aos dados serão mantidas.

Em bancos de dados relacionais, são estruturas que definem as tabelas para o armazenamento das informações. De forma análoga, são tabelas que armazenam as definições de outras tabelas (RUMBAUGH et al., 1994, p. 94).

2.3.5 *Create*, *Retrieve*, *Update* e *Delete* - CRUD

CRUD é uma sigla referente as quatro operações básicas utilizadas em banco de dados: inserção, recuperação, modificação e exclusão de informações (do inglês, *Create, Retrieve, Update e Delete*) ou seja, os métodos padrões de DML utilizados na arquitetura de banco de dados e manipulação de objetos na programação orientada a objetos. São os mais básicos métodos de uma classe.

2.3.6 Sistema de Gerenciamento de Banco de Dados MySQL

O MySQL é um SGBD considerado recente no mercado em relação a outros, porém, hoje o mesmo tem grande potencial para disputar mercado esses citados e sem dúvida é um dos SGBDs mais usados.

> O MySQL é um SGBD relacional, de licença dupla (sendo uma delas de *software* livre), projetado inicialmente para trabalhar com aplicações de pequeno e médio portes, mas hoje atendendo a aplicações de grande porte e com mais vantagens do que seus concorrentes. Possui todas as características que um banco de dados de grande porte precisa, sendo reconhecido por algumas entidades como o banco de dados *open source* com maior capacidade para concorrer com programas similares de código fechado, tais como SQL Server (da Microsoft) e Oracle. (BEM-VINDO AO MYSQL).

Foi desenvolvido durante a década de 90, como objetivo principal as aplicações *Web*. Para isso era necessária uma ferramenta com um bom desempenho, Para que esse ganho de desempenho fosse possível foi desenvolvida uma API[2], escrita nas linguagens C e C++.

Inicialmente esse banco apresentava algumas limitações, mas a partir da versão 5.0, estas limitações foram solucionados, e hoje o MySQL tem concorrido de igual para igual com as demais ferramentas do mercado.

> Uma das grandes vantagens do MySQL está na licença de uso. Trata-se de uma ferramenta de código fonte aberto (*Open Source*), basicamente ela possui duas formas de uso uma software livre, que é baseada nas cláusulas GNU-PL (*General Public Licence*) e outra para uso comercial. Sobre a licença comercial, existe para algumas situações em que é necessário embutir MySQL em aplicações comerciais, ou suporte específico, ou mesmo aquisição de outras ferramentas de apoio.
> O MySQL é um SGBD portável a diversos Sistemas Operacionais, onde se destacam Linux, Unix, FreeBSD, Mac OS X Server, Windows e linguagens de programação como Java, Python, PHP, Perl, C, C++, entre outras.
> Sobre a licença comercial, existe para algumas situações em que é necessário embutir MySQL em aplicações comerciais, ou suporte

2 Interface de Programação de Aplicativos (do inglês, *Application Programming Interface*) é um conjunto de rotinas, protocolos e ferramentas para a construção de aplicações de *software*.

> específico, ou mesmo aquisição de outras ferramentas de apoio.
> O MySQL é um SGBD portável a diversos Sistemas Operacionais, onde se destacam Linux, Unix, FreeBSD, Mac OS X Server, Windows e linguagens de programação como Java, Python, PHP, Perl, C, C++, entre outras. (ALMEIDA, 2007).

Para este trabalho optou-se por usar o banco de dados MySQL.

2.4 LINGUAGENS DE PROGRAMAÇÃO

"Para se implementar um algoritmo em um computador, é necessário descrevê-lo de uma forma que o computador esteja apto a executá-lo. Essa descrição é feita por intermédio de uma linguagem de programação" (GUDWIN,1997). As linguagens de Programação funcionam como um elo entre a linguagem de máquina e os desenvolvedores, que não é nem um pouco amigável ao programador e demanda de um grande esforço na elaboração de programas mais complexos.

“Linguagens de programação, pela sua própria natureza, são criadas e mudam muito rapidamente. Cada nicho, necessidade e mercado criam oportunidades para o desenvolvimento de novas formas”. ”Embora a evolução das linguagens permaneça aparentemente restrita, uma tendência atual pode ter um impacto positivo no desenvolvimento e aperfeiçoamento dos conceitos de programação” (FERRAZ, 2003).

De forma reduzida uma Linguagem de Programação (LP) pode ser definida como um conjunto de instruções a serem executadas pelo computador para realizar um determinado processo.

As LPs podem ser classificadas em duas: compiladas ou interpretadas. Nas compiladas, o código-fonte do programa é lido por um programa chamado compilador, que cria um arquivo binário, executável diretamente pelo hardware. Já programas escritos em linguagens interpretadas não usam compiladores e são

executados utilizando um outro programa, o interpretador. É o interpretador quem lê o código-fonte e o interpreta.

2.4.1 Evolução das linguagens de programação

No final da deca de 50 os computadores disponibilizados eram lentos, pouco confiáveis e caros, não ofereciam muita memória tornando a programação difícil devido à falta de software de apoio. A programação era feita em código de máquina, era uma tarefa tediosa e propensa a erros pois as instruções eram especificadas em código de máquina, a chegada da linguagem Assembly veio minimizar esse problema.

A partir do *Assembly* surgiram as primeiras LPs, que inicialmente eram fortemente influenciadas pela linguagem de máquina. O foco era a eficiência computacional, porque os recursos de memória e processadores eram escassos.

No final da década de 60 os recursos computacionais se desenvolveram e os computadores iam se tornando cada vez mais poderosos e úteis. A partir dai as LPs passaram a focar a produtividade dos programadores, surgindo LPs que realçavam a programação estruturada.

No final da década de 70 ouve mais uma grande mudança nas LPs, ouve um aumento na complexidade dos sistemas computacionais, o que trouxe a necessidade de uma abstração de dados como técnica de programação.

Durante os anos 80 e 90 ouve a disseminação do uso de computadores pessoais e estações de trabalho. Surge então a indústria de *software* e, com ela, a necessidade de se produzir e atualizar *software* rapidamente. O reuso passa a ser um conceito central para a produtividade no desenvolvimento de *software* e para atender a esse último requisito são desenvolvidas as LPs orientadas a objetos. (ALMEIDA,2010).

2.4.2 Linguagens de programação para a *web*

A *Internet* vem passando por uma evolução muito acelerada desde sua explosão, os sites que inicialmente eram desenvolvidos no formato HTML, possuindo apenas conteúdo estático, foi deixada para trás, trazendo a necessidade de novas LPs voltadas para esta área.

> Nem sempre a *Internet* foi dinâmica, eficiente e útil como é atualmente. Em seus primórdios era estática, feia e com pouca utilidade para nós que a utilizamos como fonte de lazer, entretenimento e solução para vários problemas da vida moderna.
> Com o advento das linguagens de programação para a *web* e também da migração de várias linguagens que já existiam para este ambiente, começaram a serem oferecidos vários serviços, que hoje vão desde a uma simples busca de informação[..] até a compra de produtos e serviços de empresas inexistentes no mundo físico, mas que podem ser entregues na sua casa de uma forma rápida e cômoda.(BARRÉRE et al., 2010).

Surgiram então LPs como o PHP (*Hipertext Preprocessor*), ASP (*Active Server Pages*), JSP (*Java Server Pages*) entre outras. possibilitando aos desenvolvedores gerar conteúdos dinâmicos para internet muitas vezes auxiliadas pelos SGBDs trazendo cada vez mais comodidade aos usuários.

2.4.3 *Hypertext Preprocessor* - PHP

A palavra PHP é um acrônimo recursivo para *Hypertext Preprocessor* (em

tradução livre "Preprocessador Hipertexto") é uma LP interpretada, de código-fonte livre e de uso geral e muito usada na atualidade.

“O produto foi originalmente chamado de *Personal Home Page Tools* mas como se expandiu em escopo, um nome novo e mais apropriado foi escolhido por votação da comunidade. O PHP está atualmente na versão 5, chamado de PHP5 ou simplesmente de PHP.” (WELLING et al., 2005).

Segundo MELO et al. (2007) ao contrario de outras linguagens o PHP possibilita que as paginas contenham código juntamente com HTML o que permite alternações durante toda a pagina. Outra grande vantagem do PHP sobre outras linguagens é que devido ao mesmo ser uma linguagem interpretada, o código é executado diretamente no servidor, retornado ao cliente apenas o HTML (*HyperText Markup Language*), em outras palavras, o cliente recebe apenas os resultados da interpretação, mas não saberia como é o código fonte.

> Qualquer coisa que pode ser feita por algum programa CGI[3] pode ser feita também com PHP, como coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber *cookies*. PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros. Construir uma página baseada em um banco de dados torna-se uma tarefa extremamente simples com PHP. Além disso, PHP tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP. Ainda é possível abrir sockets e interagir com outros protocolos. (SANTOS, 2005).

Devido a simplicidade do uso, o PHP é uma das linguagens mais indicadas para iniciantes, mas que também oferece vários recursos avançados e de ultima geração para desenvolvedores profissionais. Devido a fatores como este, o PHP será a LP usada neste trabalho.

2.5 FERRAMENTAS CASE

3 O *Common Gateway Interface* (CGI) são programas executáveis hospedados no servidor.

Engenharia de *software* auxiliada por computador (Computer-Aided Software Engineering - CASE). São na realidade aplicativos que auxiliam os desenvolvedores no projeto e desenvolvimento de software. “Inicialmente, era utilizado o termo *Workbench*, que significa "bancada de trabalho". Ele designava as ferramentas, geralmente automatizadas, que auxiliavam no trabalho dos desenvolvedores de software. Mais tarde surgiu o termo CASE”.(Azevedo, 1998).

Segundo CALIARI (2010) cada ferramenta tem propósitos diferentes, fornece serviços diferentes, mas possuem algumas características em comum.

Uma outra definição é dada por Loh (1996): uma ferramenta CASE é qualquer *software* que auxilia as pessoas (desenvolvedores e analistas) que trabalham numa empresa. A presença de ferramentas CASE é vital hoje em dia para o bom funcionamento de uma empresa desenvolvedora de *softwares*. Elas existem auxiliando todo o ciclo de desenvolvimento (análise, projeto, implementação e teste) e são também de suma importância para a manutenção do *software*. Há também ferramentas CASE para apoiar a gerência dos projetos de desenvolvimento.

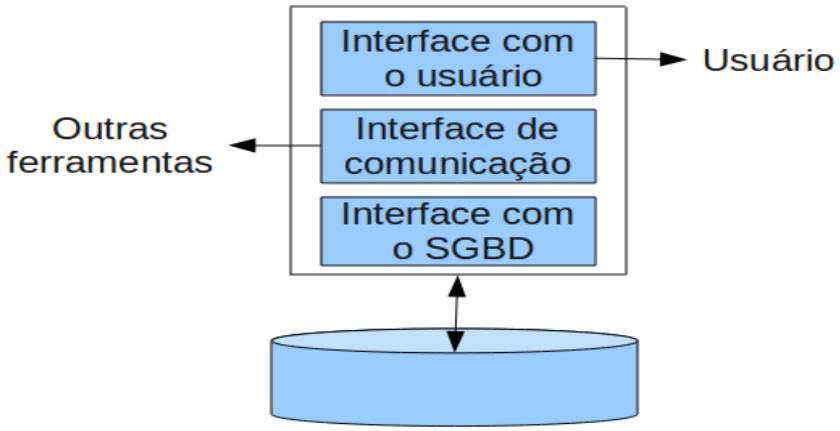

Figura 2: Composição básica de uma ferramenta CASE (CALIARI, 2010)

2.6 PADRÕES DE ARQUITETURA DE *SOFTWARE*

Segundo ROCHA (2010) “arquitetura de *software* é uma subárea da disciplina de Engenharia de *Software*, cujo objetivo é estudar os componentes do sistema, suas propriedades externas, e seus relacionamentos com outros *softwares*”.

Ainda segundo ROCHA (2010) “uma boa arquitetura pode possibilitar que um sistema satisfaça às exigências principais de um projeto, tais como: desempenho, confiabilidade, portabilidade, manutenibilidade, interoperabilidade e etc., já uma arquitetura má elaborada pode ser desastrosa”.

A arquitetura do *software* define a estrutura do *software*, que compreende os componentes com suas propriedades visíveis externamente e os relacionamentos entre eles.

A arquitetura dos *softwares* tem recebido crescente reconhecimento e atenção pois tem desempenhado um papel importante como ponte entre requisitos e implementação no processo de desenvolvimento. Existem diversas contribuições para facilitar e difundi-las.

A arquitetura de *software* ainda esta em ascendência nas áreas acadêmicas e no mercado, na qual têm sido realizados diversos estudos com o intuito de melhorar a qualidade dos produtos que são desenvolvidos.

> O objetivo principal da arquitetura de *software* é satisfazer os requisitos necessários para atender o negócio. Tais requisitos ocorrem em momentos distintos durante o desenvolvimento do sistema. Por isso, é importante saber em qual fase do processo a visão de um esquema deve ser utilizada.
> No entanto, a arquitetura deve satisfazer os requisitos necessários para o sistema. Dentre os requisitos, há aqueles que são identificados pelos participantes que são responsáveis pelo negócio e desejam uma solução computacional para o problema que enfrentam. Estes requisitos estão ligados às funcionalidades que o sistema deve atender. Porém, há outros requisitos implícitos, denominados requisitos não funcionais ou atributos de qualidade que devem ser identificados e compreendidos pelos arquitetos, analistas e projetistas de sistemas. ROCHA (2010).

Segundo WTHREEX (2010), os padrões de arquitetura são divididos em

quatro categorias: estrutura, sistemas distribuídos, sistemas interativos e sistemas adaptáveis.

Tabela 1: Categorias de padrões de arquitetura. Fonte: WTHREEX, 2010.

Categoria	Padrão
Estrutura	Camadas
	Pipes e Filtros
	Quadro-negro
Sistemas Distribuídos	Broker
Sistemas Interativos	Modelo-Visão-Controlador
	Apresentação-Abstração-Controle
Sistemas Adaptáveis	Reflexo
	Microkernel

2.6.1 Padrão modelo, visão, controlador - MVC

Consiste em um padrão de arquitetura de aplicações que visa separar a lógica da aplicação (*Model*), da interface do usuário (*View*) e do fluxo da aplicação (*Controller*). Permite que a mesma lógica de negócios possa ser acessada e visualizada por várias interfaces.

O MVC também é utilizado em padrões de projetos de *software*, entretanto, MVC abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto.

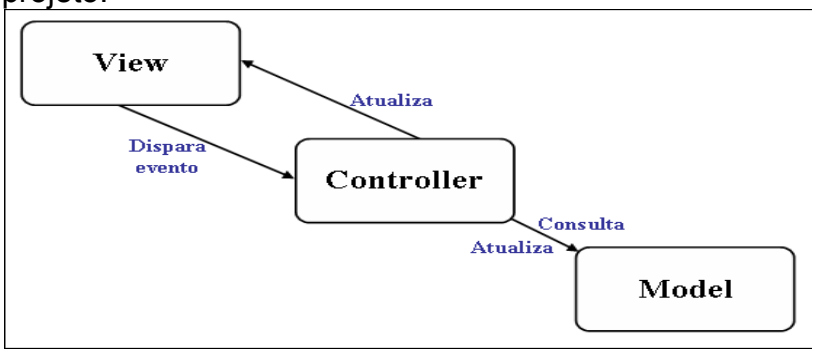

Figura 3: Padrão MVC KLUG(2007)

> A utilização de MVC para aplicações *web* faz com que se desenvolva uma aplicação bem estruturada e com módulos bem distintos. Sendo que os web designers, que são os profissionais responsáveis pela parte visual da aplicação, não precisem entender rotinas de programação que são desenvolvidas por programadores propriamente ditos.
> Para uma implementação correta, as camadas *Model* , *Controller* e *View* devem ser implementadas de forma que a inversão da ordem não acarrete problemas por dependência, ou seja a camada de interface (*View*) depende de controle (*Controller*) que implementa um Modelo (*Model*), mas nunca o inverso. (Ibarra et al.,2007).

2.7 QUALIDADE DE *SOFTWARE*

O produto *software* passa a ser, cada vez mais, um componente comum em uma série de outros produtos, desde carros, fornos de micro-ondas, elevadores, telefones, até sistemas de informação organizacionais.

Segundo BARTIÉ (2002), hoje em uma definição bem simples podemos dizer que a qualidade do *software* pode ser descrita desde a fase do projeto até a venda ao cliente, pois qualquer alteração ali existente pode alterar todo o ciclo do projeto ocasionando desde um alargamento no prazo de entrega até um cancelamento do desenvolvimento por motivos de inviabilidade econômica do projeto. Para que isso não ocorra mais com tanta frequência como era de costume das empresas de *software*, buscou-se uma adequação as tendências de qualidade que o mercado exigia.

Ainda Segundo BARTIÉ(2002), para que se torne possível obter um *software* de qualidade não podemos ter processos de desenvolvimento frágeis ou deficientes, então para que possamos estabelecer um processo de garantia da qualidade do *software* devemos enfocar o produto tecnológico e o processo de desenvolvimento.

> De um produto exige-se qualidade e preço. Portanto como produto, *software* tem que ter o nível de qualidade exigido e procurar ser

> desenvolvido no menor custo possível. Esta é exatamente a função da engenharia, procurar sistemas de melhor qualidade dentro de um custo compatível com essa qualidade, otimizando a redução de custos.
> Obter qualidade nos processos e produtos de engenharia de *software* não é uma tarefa trivial. São vários os fatores que dificultam atingir os objetivos de qualidade. No entanto, nada mais decepcionante do que produzir software que não satisfaça as necessidades dos clientes. Grande volume de recursos são gastos, mas, em muitos casos, ocorre uma grande frustração por parte dos clientes, quando se depara com a forma final apresentada pelo *software* encomendado.(LEITE, 2010).

Se um *software* com muitos erros e falhas perde parte do valor para o usuário, um *software* isento de erros não é, necessariamente, um *software* de qualidade para seu usuário, ou seja, seria o programa certo para o problema errado.

As necessidades do usuário são expressas nos requisitos explícitos, mas também nos requisitos implícitos, que normalmente não são declarados, como: flexível, fácil de operar, barato, construído no prazo etc. Ou seja, o resultado deve ser mais prático, mais rápido, mais fácil, mais seguro e mais barato em relação ao processo convencional usado antes da informatização.

A ausência de funções necessárias, não pode ser compensada por funções auxiliares genéricas não solicitadas, como: calculadora, agenda, cor de tela etc. Um *software* de qualidade deve satisfazer as necessidades do cliente e não somente funcionar direito e não ter erros. *Software* de qualidade é aquele que, não apenas satisfaz as exigências, mas também é implementado a tempo e de acordo com o orçamento.

> A melhoria da qualidade em *software* requer: Projeto realizado dentro de um rigor científico, buscando dar qualidade ao produto de *software*; Controle do processo de desenvolvimento; Medição do processo de desenvolvimento; Técnicas de Garantia da Qualidade; Atividades auxiliares independentes de fase (gerenciamento de configuração, técnicas de melhoria contínua) e Utilização de Métodos, Padrões e Ferramentas adequadas. (PORTELLA, 2006).

2.7.1 Qualidade do processo

Segundo MASETO (2006) “para poder garantir a qualidade do *software*, precisamos estruturar processos que possuam mecanismos de inibição e impedimento de falhas, possibilitando a identificação prematura de defeitos. Nisto devemos incluir todas as partes do processo de desenvolvimento, desde os requisitos levantados até as projeções financeiras. Ou seja, todos e quaisquer documentos gerados durante o processo de desenvolvimento”.

2.7.2 Qualidade do produto

A qualidade do produto é baseada principalmente sobre os testes aplicados sobre eles. Cada empresa trabalha com seus próprios métodos empregando os tipos de testes que mais acharem necessários. A tática mais comum é usar uma fase especifica no projeto para a execução dos testes.

Segundo MASETO (2006), estes têm tendência a apresentar um grau de eficiência muito baixo, o que faz muitas empresas substituir este processo. Para que isso não ocorra, ele aponta quatro passos: fazer um planejamento maior sobre os testes, testar as funcionalidades antigas e atuais, criar mecanismos de automação e fazer conferência com a equipe sobre os testes.

2.8 CONCEITOS DE AUTOMATIZAÇÃO E AUTOMAÇÃO

Quando o assunto é automação e automatização, surgem diversas dúvidas a

respeito do significado de cada uma e de como devem usadas.

O conceito de automação é tornar automáticas atividades repetitivas com uso de sistemas e equipamentos que efetuam coleta de dados e atuam nos processos, minimizando a necessidade da interferência humana e resultando em maior velocidade nas operações, redução de erros, redução de custos, controle e principalmente em fidelidade de informações, elementos essenciais para um gerenciamento eficaz.

Automatizar é obter um melhor gerenciamento operacional em todas as áreas da empresa, inclusive em seu relacionamento com parceiros comerciais e clientes. Hoje em dia o varejista deve acompanhar as transformações que estão acontecendo e fazer da tecnologia sua aliada inseparável. O momento atual da economia exige que as empresas reorientem suas estratégias à necessidade do consumidor. É prioritário conhecer os hábitos de consumo desses, e estar atento a mudanças nos mesmos.

Com a automação em pleno funcionamento, o varejista terá tempo para se tornar mais estratégico e poderá dirigir sua atenção à capacitação profissional de seu pessoal, aos seus clientes, ao seu negócio.

Automatização vem de ato ou efeito de automatizar. Inicialmente, podemos definir adequadamente o que vem a ser automação de processos e qual a diferença, se há, entre automação e automatização.

> Uma mudança tecnológica que substitui pessoas por máquina. Começou na Revolução industrial e continua como uma opção de mudança hoje. Exemplo de automação é a introdução de triagem automática de correspondência pelo serviço de Correios do EUA. e robôs nas linhas de montagens de automóveis. Robbins (1999).

Robbins (1999) utiliza os termos automação e automatização como sinônimos. A semelhança entre os termos é mais bem expressa por Arnold & White:

> As palavras automáticas, autômato e mesmo automatismo já há muito constam do nosso idioma, porém a palavra automação (ou automatização)

> só há pouco penetrou em nossos dicionários. Em essência, pouca diferença há no significado das palavras automatizado e automático; essa diferença reside, principalmente, na extensão do automatismo.(ROBBINS,1999).

Para VILARINHO(2010), “é possível ainda, encontrar na literatura, o termo automação sendo empregado, atualmente, para definir o processo de inovação tecnológica de base microeletrônica. É com este significado que se nomeiam, por exemplo, os processos de automação bancária ou automação industrial, traduzindo a utilização da informática nesses setores. O significado do termo é, no entanto, bem mais amplo. Ele diz respeito a todo instrumento ou objeto que funcione sem a intervenção humana, podendo ser aplicado a qualquer tipo de máquina ou artefato que opere desse modo”.

Assim como definido anteriormente, esses termos aparecem como sinônimos, e implicam a substituição de trabalho humano por trabalho automatizado, ou, tal como no foco deste trabalho, a substituição do homem pela máquina em tarefas repetitivas e rotineiras.

2.9 FRAMEWORKS

De acordo com MINETTO (2007), um *framework* de desenvolvimento é uma base de onde se pode desenvolver algo maior ou mais específico. É uma coleção de códigos-fonte, classes, funções, técnicas e metodologias que facilitam o desenvolvimento de novos *softwares*.

> Pode-se separar o *framework* em partes que ficarão estáticas, desempenhando as suas funções sem modificações, das partes que poderão/deverão ser modificadas para prover uma funcionalidade especial. A parte imutável é comum ao domínio ao qual o *framework* serve e não sofre alterações quando é aplicada em sistemas. Já a parte flexível não representa uma solução completa, pois permite que o desenvolvedor a customize para adequá-la a sua aplicação. (BITTENCOURT DE SOUZA,

2004).

O reuso de *software* tem sido um dos principais objetivos da engenharia de *software*. Reutilizar *software* não é simples. Com o surgimento do paradigma da orientação a objetos, a tecnologia adequada para reuso de grandes componentes tornou-se disponível e resultou na definição de *frameworks* orientados a objetos.

Os *frameworks* têm atraído a atenção de muitos pesquisadores e engenheiros de *software* e têm sido definidos para uma grande variedade de domínios. As principais vantagens de um *framework* são o aumento do reuso e a redução do tempo para desenvolvimento de aplicações. *Frameworks* são uma forma particular de representar arquiteturas, embora existam arquiteturas que não podem ser representadas como *frameworks*.

Os principais benefícios dos *frameworks* orientados a objetos decorrem da modularidade, reusabilidade, extensibilidade e inversão de controle que eles oferecem aos desenvolvedores. As vantagens do uso variam entre agilidade, e padronização, aproveitamento e compatibilidade e não há necessidade de se implementar partes básicas de um sistema, isso já vem pronto no *framework*, isso agiliza muito o processo de desenvolvimento. Uma desvantagem é que a maioria dos *frameworks* te obrigam a desenvolver segundo sua filosofia, e a adaptação a essa filosofia pode ser um pouco difícil e em relação ao desenvolvimento tradicional trás menor desempenho.

Frameworks aumentam modularidade através do encapsulamento de detalhes voláteis de implementação por trás de interfaces estáveis. A modularidade dos *frameworks* ajuda a aumentar a qualidade do *software*, concentrando o impacto das mudanças de projeto e implementação, o que reduz o esforço necessário para entender e manter *softwares* existentes.

Utilizando um *framework* ganha-se agilidade no desenvolvimento, entretanto todos os *frameworks* tem seus pontos fracos, um dos mais comuns é a customização de interfaces e componentes.

2.9.1 Diferença entre biblioteca, *toolkit*, *framework*, *engine* e API

Existe uma grande dificuldade no momento de distinguir bibliotecas, *toolkit*, *framework* e API, abaixo segue algumas diferenciações entre os mesmos.

Uma biblioteca pode ser entendida como um conjunto de código pronto. Pode ser um conjunto de funções não necessariamente orientada a objetos. Portanto, mesmo as *frameworks* são compostos por bibliotecas.

Um *toolkit* é um conjunto de classes, mas cuja arquitetura não se baseia em extensão, mas sim em uso. Normalmente são implementados como uma biblioteca de rotinas ou uma plataforma para aplicativos que auxiliam numa tarefa. Segundo MASETO (2006) "*toolkit* é um conjunto de elementos básicos para construção de software. Normalmente são implementados como uma biblioteca de rotinas ou uma plataforma para aplicativos".

Framework são bibliotecas de classes trabalhando para dar suporte a uma funcionalidade. O *framework*, é mais intrusivo e gera mais acoplamento do que um *Toolkit*. Geralmente, um *framework* chega a impor algum tipo de arquitetura ao projeto de *software*. O *framework* se diferencia pois a biblioteca de classes se concentra em apenas oferecer implementação de funcionalidades, sem definir a reutilização de uma solução de arquitetura.

Engines são *frameworks* que na maioria das vezes gerencia o fluxo do código.

Uma Interface de Programação de Aplicativos (do inglês, *Application Programming Interface*) funciona como uma uma conexão entre dois pontos: Interface com o usuário e Interface de programação de aplicações.

Segundo o *Software Engineering Institute* (SEI)[4], uma API é uma tecnologia que facilita a troca de mensagens ou dados entre duas ou mais aplicações.

4 http://www.sei.cmu.edu/str/descriptions/api.html

Inicialmente, APIs eram tecnologias baseadas apenas em simples chamadas de sub-rotinas. Entretanto, elas foram expandidas e hoje incluem novas funcionalidades e permitem interoperabilidade e mudanças em sistemas, permitindo compartilhamento de dados entre diversas aplicações.

Segundo SOUZA (2010) na prática, uma API significa ou indica uma interface que define um conjunto de serviços que um componente, módulo, ou aplicação fornece a outros elementos de software.

2.9.2 *Frameworks* PHP

Ao se analisar os principais *frameworks* de desenvolvimento para PHP, nota-se que algumas tecnologias e técnicas são comuns à maioria deles modificando apenas a forma como como são organizados.

O uso de *framework* traz velocidade e organização no processo do desenvolvimento tornando o trabalho menos cansativo com relação ao desenvolvimento tradicional.

> Os frameworks estão presentes há vários anos. O primeiro framework para web que Rob usou em um projeto verdadeiro foi o Fusebox, que foi originalmente escrito para ColdFusion. Vários outros frameworks apareceram desde então, sendo o Struts o próximo de maior destaque, escrito em Java. Vários clones PHP do Struts foram escritos, mas não foram muito bem traduzidos para o PHP. O maior problema estava no fato de que as aplicações web em Java são executadas em uma máquina virtual que executa continuamente, de modo que o tempo de inicialização da aplicação web não é relevante em cada solicitação web. O PHP inicializa cada solicitação a partir de um estado anterior novo, de modo que a longa inicialização exigida pelos clones do Struts os tornaram relativamente lentos enquanto resultado.
> Alguns anos atrás, um novo framework, chamado Rails, surgiu baseado em uma linguagem relativamente desconhecida chamada Ruby. Rails (ou Ruby on Rails, como ela também é conhecida) promoveu o conceito de convenção sobre configuração e provocou uma revolução no mundo do

> desenvolvimento para web. Logo depois que o Rails surgiu, vários clones diretos do PHP apareceram, juntamente com diversos frameworks inspirados no Rails, em vez de cópias diretas.(ALLEN et al.,2009).

Entre os *frameworks* utilizados na atualidade podem ser citados: CakePHP, CodeIgniter e Zend.

2.9.2.1 CakePHP

CakePHP é um *framework* de desenvolvimento rápido para PHP que fornece uma arquitetura extensível para desenvolvimento, manutenção, e distribuição de aplicações. Usando *design patterns* (padrões de projeto) conhecidos como MVC e ORM com convenção sobre o paradigma da configuração, reduz o custo do desenvolvimento e ajuda os desenvolvedores a escreverem menos código. (CAKEPHP, 2010).

Tem como principais vantagens:

- Comunidade ativa e amigável.
- Licença flexível.
- Compatibilidade com PHP 4 e PHP 5.
- Integrando funcionalidade CRUD para interagir com o banco de dados.
- Aplicações scaffolding.
- Geração de código.
- Arquitetura MVC.
- Requisições ao expedidor com clareza, URLs personalizáveis e rotas.
- Facilita o uso de AJAX, JavaScript, HTML, formulários e outros nas

visões.

- Internacionalização.
- Funciona em qualquer subdiretório de um servidor, com poucas configurações no Apache.

CakePHP é um *framework* em PHP gratuito, de código aberto, para desenvolvimento ágil. Possui uma ótima estrutura para programadores criarem aplicações web. O principal objetivo do CakePHP é permitir o trabalho sobre uma estrutura de forma que o desenvolvedor possa programar de forma rápida e sem a perda de flexibilidade.

2.9.2.2 CodeIgniter

O Objetivo principal do CodeIgniter é possibilitar que o desenvolvedor crie projetos mais rápido que se estivesse codificando do zero usando um abrangente conjunto de bibliotecas, minimizando a quantidade de código necessário para uma determinada tarefa.

> CodeIgniter é um *framework* PHP desenvolvido sob o paradigma da orientação a objetos e o padrão MVC. Foi inspirado no *Ruby* on *Rails* e é um dos mais bem documentados *frameworks* PHP disponíveis atualmente.
>
> Com o CodeIgniter, você ganha muito em agilidade. Com o *framework* é muito simples fazer reaproveitamento de código e acessar métodos e funções previamente construídos. Você terá acesso a bibliotecas de funções PHP e ajudantes que automatizarão tarefas rotineiras e a elaboração de código XHTML válido.(GABARDO, 2010).

2.9.2.3 Zend *Framework*

Segundo ALLEN et al. (2009) o Zend *Framework* é uma biblioteca PHP para o desenvolvimento de aplicações web com PHP. Os componentes se encaixam para oferecer um *framework* completo com todos os componentes necessários para construir aplicações modernas, fáceis de serem criadas e mantidas.

ALLEN et al.(2009) diz ainda que este é composto por vários componentes distintos que podem ser agrupados em seis categorias de mais alto nível. Possuindo tudo o que é necessário para desenvolver aplicações *web* profissionais. No entanto, o sistema é bastante flexível e foi criado para permitir a escolha das partes do *framework* que são aplicáveis à todas as situações.

O Zend *Framework* é escrito sobre o PHP5 orientado a objetos usando técnicas modernas de padrões de projeto (*design patterns*), desta forma, ele é modular, o que faz com que cada componente não dependa de vários outros componentes. Além disso, este *framework* possui uma documentação completa e de fácil acesso.

2.10 *SCAFFOLDING* E GERADORES DE CÓDIGO

O *scaffolding* provê uma estrutura básica para operações CRUD, ou seja, aquelas operações básicas que temos na manipulação de dados, gerando interfaces rápidas que podem apoiar o desenvolvimento até que você insira a codificação necessária.

O recurso de scaffolding de aplicações é uma técnica que permite ao desenvolvedor definir e criar uma aplicação básica que manipula os dados do banco de dados.

Segundo BROWN (2008) os sistemas de *scaffolding* apresentam como benefícios a criação de sistemas administrativos, que muitas vezes possuem

dezenas de listagens de tabelas de banco, todas muito similares. Ou seja, uma boa parte do tempo de desenvolvimento e testes são investidos em criação partes básicas. O potencial de erros é elevado exponencialmente. O *scaffolding* elimina quase que por inteiro este processo. O sistema gera a grande parte deste código para o desenvolvedor. Isso reduz o risco de propagação de um erro, ou erros por digitação, mantém os sistemas padronizados, e assim por diante.

Por outro lado, uma vez gerados os códigos, implementações de mudanças são complicadas. Adicionar campos em uma tabela, modificar relacionamentos entre as mesmas podem influenciar a funcionalidade do sistema. Modificar essas classes geradas dinamicamente é bem complicado. Se os códigos forem dinamicamente gerados novamente, corre-se o risco de perder as modificações que feitas, tendo em vista que o *software reescreve* os arquivos.

O uso de *scaffolding* poupa o trabalho da criação da estrutura real para acelerar o início de um projeto em etapas iniciais. É uma excelente maneira de iniciar o desenvolvimento de partes prematuras de uma aplicação.

Geradores de código e scaffolding são considerados sinônimos, pois possuem as mesmas características.

Geradores de código são basicamente programas que geram outros programas. Os geradores podem ser definidos como ferramentas tanto para formatar códigos simples quanto para gerar aplicações complexas a partir de modelos abstratos (*templates*). São muito utilizados para agilizar o processo de desenvolvimento, pois aumentam a produtividade e diminuem o tempo gasto na codificação da aplicação e, consequentemente, o custo final (AMBLER, 2004; KLUG, 2007).

2.11 DESENVOLVIMENTO RÁPIDO DE APLICAÇÃO

Desenvolvimento Rápido de Aplicação (ou *Rapid Application Development,* em inglês) (RAD) é um termo usado originalmente descrever a processo do desenvolvimento do *software* introduzido perto James Martin em 1991. A metodologia de Martin envolve o desenvolvimento iterativo e a construção de protótipos.

Mais recentemente, o termo e seu acrônimo vieram ser usados em um sentido mais largo, genérico que abrangesse uma variedade das técnicas visadas apressando o desenvolvimento de uma aplicação. As aproximações do RAD podem envolver acordos na funcionalidade e desempenho na troca para permitir um desenvolvimento mais rápido e facilitar a manutenção da aplicação.

Com o uso do RAD tem-se a possibilidade do reuso de componentes, com isso pode-se economizar recursos. No desenvolvimento é feito em um nível mais alto com grande abstração. Como trabalha com prototipagem há um maior envolvimento do usuário, o que pode vir a reduzir necessidades de manutenção.

O RAD é apropriado quando o escopo do projeto é restrito, ou seja, o objetivo do projeto já esta determinado, sendo que o mesmo possa ser dividido em vários módulos independentes.

3 METODOLOGIA

Os próximos capítulos descrevem o motivo pelo qual foi escolhido o *scaffolding*, comparação do método em relação aos *frameworks*, e o caminho percorrido para chegar a um resultado satisfatório descrevendo as tecnologias aplicadas e como usá-las.

3.1 DESENVOLVIMENTO COM BANCO DE DADOS

Segundo SOUZA (2007) os bancos de são dados têm uma grande utilização por meio das aplicações. Praticamente toda interação entre um usuário e o SGBD se dá, de forma transparente, por meio de uma aplicação, para isso todo sistemas de banco de dados fornece suporte às ferramentas de desenvolvimento de aplicações.

Seu uso atualmente, esta explicito em todas as áreas de desenvolvimento que necessitam de tratamento de dados. Todos os *frameworks* da atualidade tornam o uso do SGBD quase que relevante, pois possuem bibliotecas que fazem interação dos SGBDs mais usados com as aplicações.

Atualmente um desenvolvedor de sistemas tem duas possíveis reações diante de um projeto que faça uso de banco de dados em um curto espaço de tempo, caso o mesmo tenha conhecimento de algum *framework*, mesmo que superficial, vê a codificação do projeto de forma diferente de um desenvolvedor que não possua tal conhecimento. O primeiro usa seu conhecimento a respeito do *framework* para verificar se o mesmo é possibilita que o o mesmo trabalhe com *scaffolding* ou algo similar. Já o segundo pode optar em aprender sobre algum *framework*, mas

possivelmente devido a escassez de tempo irá desenvolver o projeto de forma tradicional, ou seja, codificando manualmente todo o código.

O primeiro tem mais chances de terminar o projeto em menor tempo que o segundo, pois o mesmo já terá toda a base de códigos do *framework* para auxiliá-lo no desenvolvimento, restando apenas a tarefa de alterá-los até encontrar o resultado desejado. O segundo terá que fazer todas as analises para o código e mesmo usando alguns recursos como bibliotecas de classes ou alguma API, terá uma certa dificuldade pois terá que criar no mínimo quatro códigos diferentes para cada tabela do banco de dados para que possa fazer operações básicas com dados nas mesmas. Portanto, nestes casos há necessidade de uma ferramenta de auxilio para geração de códigos para estes desenvolvedores.

3.1.1 *Frameworks* e *scaffolding*

Muitos desenvolvedores abordam como desvantagem o uso de *frameworks* para alteração ou mesmo ampliação de sistemas legados, devido ao uso de códigos estruturados, ou tecnologias consideradas antigas, uma das soluções para resolver o problema é o uso de *scaffolding*, afinal nem sempre estes sistemas permitem alterações bruscas em sua estrutura. Podem apresentar também incompatibilidade com o *framework* que deseja-se trabalhar devido a tecnologia ultrapassada do sistema e a tecnologia considerada recente dos *frameworks*. Mesmo quando não ocorre os problemas citados, há uma grande chance de ocorrer redundância de dados ou códigos dentro do sistema, muitas vezes caracterizada como improviso.

Atualmente existem diversos frameworks que possibilitam o uso de scaffolding como, por exemplo, o CakePHP, CodeIgniter e o Zend *Framework*, porém para trabalhar com este recurso, é necessário ter um conhecimento mínimo a respeito de configuração, programação orientada a objetos (já que os mesmos são

desenvolvidos seguindo este padrão), padrões de arquitetura MVC e outras tecnologias usadas nos mesmos.

Mesmo com toda flexibilidade, os *frameworks* possuem suas limitações com relação a geração de código: o resultado sempre apresenta dependência do *framework*, pode não disponibilizar o código físico, isto é, não o armazena, é gerado e executado, sendo que o desenvolvedor ainda corre o risco de perder alterações feitas nos arquivos, caso seja necessário gerar algum código novamente. Com isso, caso o desenvolvedor queira tornar este código independente do *framework* com outros padrões e tecnologias, terá que fazer uma revisão detalhada de cada arquivo adaptando-os ao novo ambiente.

Fica claro que nestes e outros casos o uso de *frameworks* podem até apressar até certo ponto, mas depois passam a atrasar o desenvolvimento. Portanto há uma necessidade de uma ferramenta de *scaffolding* para estes casos, que em contrapartida, possibilitará ao desenvolvedor com poucos conhecimentos a respeito de *framework* desenvolver rapidamente seu trabalho.

3.2 DEFINIÇÕES DE PADRÕES E MANIPULAÇÕES DE DADOS

Para criar o gerador de códigos foi necessário definir de onde virão os dados, como serão tratados e como obter códigos a partir disso. Abaixo é definido um padrão para metadados e como organizar os metadados recuperados do SGBD MySQL. Também é definido um *template* para que a “mágica” do scaffolding aconteça.

3.2.1 Definição de um padrão para armazenamento de metadados

Os SGBDs possuem dados muitas vezes usada apenas por eles, estas informações trazem todas as informações a respeito dos dados armazenados no banco de dados, como o nome e descrição de suas tabelas, campos e suas relações. O MySQL disponibiliza um catalogo com algumas destas informações chamado de "*information_schema*" (ou esquema de informações em português). Este catalogo é representado como um banco de dados que possui varias tabelas com todas as informações a respeito da estrutura dos bancos de dados existentes.

Para que o scaffolding funcione corretamente, deve-se definir um modelo de código, que contenha expressões, variáveis e os códigos que deverão ser replicados para as tabelas, este modelo poderá ser definido da forma como o desenvolvedor preferir, nele os códigos podem ser escritos da forma como o seu criador esteja acostumado a usar a linguagem. Este vetor é importante pois é delas que sairão os dados do *scaffold.*

A única exigência para que o modelo funcione corretamente é o uso de variáveis e expressões padrões. As expressões e variáveis do scaffolding serão definidas no código como uma linguagem de marcação que diferencia os blocos de códigos, blocos de códigos condicionais do resto do código dentro do modelo.

As variáveis do modelo poderão assumir valores como o nome do banco de dados, nome da tabela, qualquer dado a respeito campo ou outra variável desde que a mesma seja reconhecida pelo gerador. Já as expressões serão usadas para que sejam supridas as necessidade de gerar um código diferenciados para campos. As expressões poderão ser divididas em duas: a expressão com função replicativa e a expressão condicional, sendo ainda possível o uso das duas. Dentro das expressões pode fazer o uso livre das variáveis do gerador, o que possibilita inúmeras expressões diferentes.

O modelo poderá ser reutilizado desde que as tecnologias façam o uso das mesmas tecnologias e padrões isto certamente diminuirá a necessidade de alterações nos códigos gerados. Estas informações necessitam de um tratamento

para que sejam organizadas em um vetor multidimensional mantendo toda a estrutura e integridade do banco de dados, este vetor deverá apresentar informações claras, Definiu-se então um modelo que pode ser representado da seguinte forma:

- O vetor apresentará uma forma de arvore;
- A raiz da arvore será sempre o nome do BD;
- O primeiro nível da arvore conterá as tabelas do BD;
- O segundo nível conterá a listagem dos campos das tabelas;
- O terceiro nível conterá informações a respeito dos campos;
- Poderá ainda aparecer um quarto nível com informações relacionadas a campos que se referenciam em campos de outra tabela, estes campos são chamados de chaves estrangeiras (*foreign keys*).

Este modelo é representado com toda sua definição abaixo:

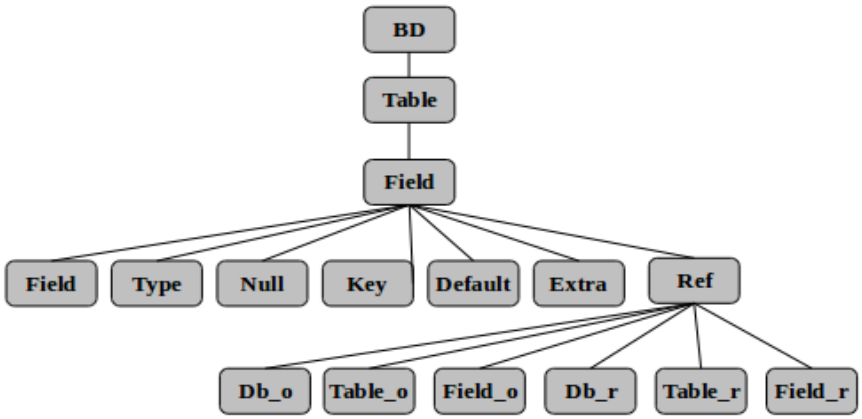

Figura 4: representação do modelo de padronização de metadados

Onde:

- BD – banco de dados trabalhado;
  - *Table* – tabela trabalhada;
    - *Field* – campo trabalhado no momento;
      - *Field* – nome do campo;

- *Type* – tipo do campo (Ex.: int, varchar, float,etc.);
- *Null* – dado que determina se o campo pode ser vazio. Assume valores de *YES* ou *NO;*
- *Key* – dado que determina se um campo é um campo chave, chave estrangeira ou índice, campo único, respectivamente PRI, MUL,UNI;
- *Default* – Determina um valor padrão para o campo;
- *Extra* – opções extras como se o campo é auto incrementável;
- *Ref* – Caso o campo seja uma chave estrangeira este campo surge para guardar as relações;
  - Db_o – banco de dados de origem;
  - Table_o – tabela de origem;
  - Field_o – campo de origem;
  - Db_r – banco de dados da chave estrangeira;
  - Table_r – tabela da chave estrangeira;
  - Field_r – nome do campo;

3.2.2 Recuperação, padronização de metadados

Com os metadados encaixados no padrão, fica fácil a manipulação das informações a respeito do banco de dados uma vez que para referenciar ao dado basta seguir o caminho lógico da raiz do modelo até ele.

O MySQL possibilita buscar tais informações de tabelas e campos através de *query* SQL, para que as informações sejam disponibilizadas de acordo com o padrão

deve-se recuperar inicialmente uma lista com as tabelas do banco de dados, depois é feita uma busca de informações a respeito dos campos de cada tabela mantendo a organização de acordo com o modelo. O MySQL não fornece uma forma direta para identificar as referências de chaves estrangeiras, portanto é necessário criar uma consulta que o faça. As consultas relacionadas aos passos acima são:

- SHOW TABLES;
- SHOW FIELDS FROM tabela;
- SELECT ref.TABLE_SCHEMA db_o, ref.TABLE_NAME table_o, ref.COLUMN_NAME field_o, ref.REFERENCED_TABLE_SCHEMA db_r, ref.REFERENCED_TABLE_NAME table_r, ref.REFERENCED_COLUMN_NAME field_r FROM information_schema.KEY_COLUMN_USAGE as ref WHERE ref.TABLE_SCHEMA=bd AND ref.TABLE_NAME=tabela AND ref.COLUMN_NAME=campo

Onde “bd”, “tabela” e “campo” podem assumir valores diferentes de acordo com o banco trabalhado.

Com este vetor em mãos é possível identificar relações entre os campos das tabelas ou mesmo descobrir que tipo de tratamento deve-se fazer ao campo. A representação deste vetor na forma de uma arvore torna-se uma ótima maneira de se analisar a estrutura de um banco de dados.

3.2.3 Definição de um *template* para a geração de código

Para que o *scaffolding* funcione, deve-se definir um *template* de código, este *template* será igual para todas as tabelas do banco de dados, mantendo um padrão nos arquivos. Ele poderá conter expressões, identificadores e variáveis, que irão

definir como e qual código será gerado para cada tabela.

O *template* poderá ser definido da forma como o desenvolvedor preferir, nele os códigos podem ser escritos da forma como esteja acostumado a programar.

A única exigência para que o modelo funcione corretamente é o uso de padrões de variáveis, identificadores e expressões. As expressões, identificadores e variáveis são definidas no código do gerador como uma linguagem de marcação que diferencia os blocos de códigos, blocos de códigos condicionais do resto do código dentro do *template*.

As variáveis do *template* poderão assumir valores como o nome do banco de dados, nome da tabela, nome do campo ou qualquer dado relacionado a ele e qualquer outra variável desde que a mesma seja definida no gerador.

As expressões serão usadas para que seja suprida a necessidade de gerar um código diferenciado para um campo ou mais. Elas poderão ser divididas em duas categorias: a expressão replicativa e expressão condicional. Dentro das expressões pode fazer o uso livre das variáveis do banco de dados e do gerador, o que possibilita inúmeras expressões diferentes.

A utilização de *templates* possibilita que a formatação do código gerado esteja externa ao código da aplicação que o gera. Dessa forma, caso exista a necessidade de alguma alteração na saída, não é preciso alterar a codificação do sistema, mas somente o arquivo utilizado como *template*. (KLUG, 2007).

O *template* poderá ser reutilizado para vários projetos desde que as tecnologias façam o uso das mesmas tecnologias e padrões isto certamente diminuirá a necessidade de alterações nos códigos gerados.

3.2.4 Aplicação de metadados no *template*

Com os metadados padronizados e o *template* de código em mãos, é feita a combinação dos dois. Esta combinação se da na foma de substituição. A substituição é feita em três passos: identificação e processamento de expressões, identificação e processamento de variáveis, recuperação de resultados.

Durante o primeiro passo as expressões são identificadas e processadas da de acordo com sua classificação. Caso a expressão seja condicional o código delimitado pela mesma é replicado para as tabelas/campos que atenderem a condição. Se a expressão for replicativas o código delimitado dentro da mesma será replicado para todas as tabelas/campos. Dentro das expressões replicativas pode também existir expressões condicionais o que possibilita mais dinamismo aos *templates*.

No segundo passo são identificadas as variáveis, cada variável encontrada é substituída pelo seu equivalente. Neste passo o gerador já possui o código gerado para todas as tabelas.

O resultado dessa substituição dos metadados são arquivos de códigos gerados dinamicamente obedecendo rigidamente ao *template*.

Os resultados são armazenados em um vetor de dados multidimensional que mantem a relação do código gerado com o banco de dados e a tabela. Optou-se por fazer o armazenamento desta forma, pois torna mais fácil a recuperação dos resultados. Desta forma a recuperação dos códigos é feita usando os nomes do banco de dados e da tabela e a partir dai serem ou não transferidos para arquivos.

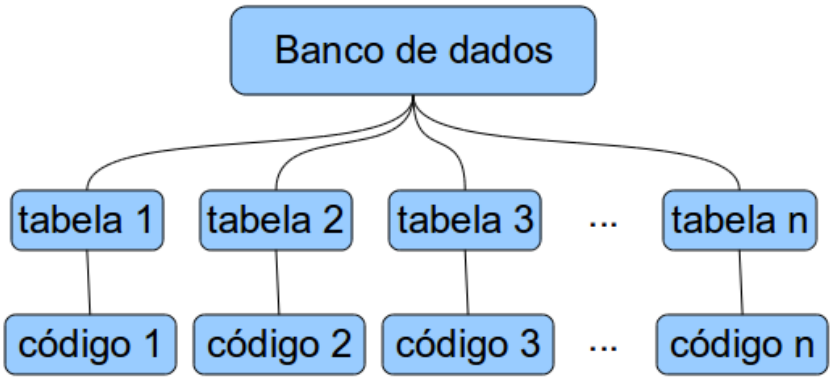

Figura 5: Representação do vetor de códigos resultantes do *scaffolding*

3.3 RESULTADOS

Para facilitar a compreensão foi feito um exemplo usando a ferramenta final. Este capitulo destina-se a exemplificação de cada parte do projeto.

3.3.1 Caso de uso

Foi tomado como caso de uso um exemplo simples de postagem de texto que possui as tabelas com seus respectivos campos[5]:

- autores: codigo, nome, email;

5 O MySQL aceita acentos em definições de nomes de tabelas e campos porém o uso não é aconselhado. Devido a este fator existem campos cujo nome contem acento, porém o mesmo não se encontra presente. Ex.: código é referenciado como codigo sem acento.

- postagem: codigo, texto, autor e situacao.

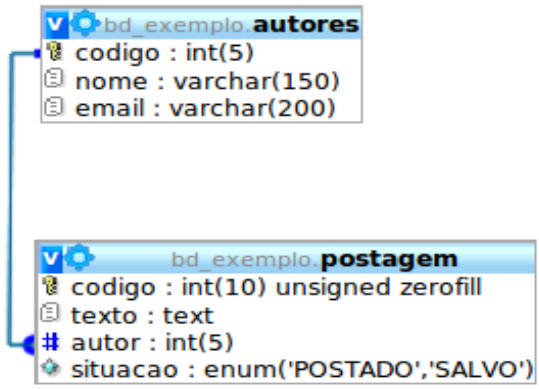

Figura 6: Estrutura do banco de dados do caso de uso

Onde o campo autor se relaciona com a tabela autores, desta forma será possível exemplificar todas as propriedades básicas do gerador. Para o caso de uso não serão apresentados todos os *templates* e códigos-fonte necessários para que o sistema funcione corretamente, serão apresentados apenas *templates* e códigos simples que exemplifiquem o funcionamento do gerador.

3.3.2 Vetor de metadados padronizados do caso de uso

O banco de dados anteriormente descrito na Figura 6 gera a tabela abaixo, extraindo todos os dados possíveis e encaixando no padrão adotado.

Pode-se notar que de acordo com a Figura 6 o campo autor da tabela postagem gera uma referência com relação ao campo codigo da tabela autor, deixando claro a dependência da tabela postagem em relação à tabela autores.

Tabela 2: Representação do vetor de metadados do caso de uso em uma tabela

bd_exemplo
autores

codigo	*Field*	codigo	
	Type	int(5)	
	Null	NO	
	Key	PRI	
	Default	NULL	
	Extra	*auto_increment*	
nome	*Field*	nome	
	Type	varchar(150)	
	Null	NO	
	Key	NULL	
	Default	NULL	
	Extra	NULL	
email	*Field*	email	
	Type	varchar(200)	
	Null	YES	
	Key	NULL	
	Default	NULL	
	Extra	NULL	
postagem			
codigo	*Field*	codigo	
	Type	int(10) unsigned zerofill	
	Null	NO	
	Key	PRI	
	Default	NULL	
	Extra	auto_increment	
texto	*Field*	texto	
	Type	text	
	Null	YES	
	Key	NULL	
	Default	NULL	
	Extra	NULL	
autor	*Field*	autor	
	Type	int(5)	
	Null	NO	
	Key	MUL	
	Default	NULL	
	Extra	NULL	
	Ref	db_o	bd_exemplo
		table_o	postagem

		field_o	autor
		db_r	bd_exemplo
		table_r	autores
		field_r	codigo
situacao	*Field*	situacao	
	Type	enum('POSTADO','SALVO')	
	Null	NO	
	Key	NULL	
	Default	NULL	
	Extra	NULL	

3.3.3 *Template* desenvolvido do caso de uso

O Quadro 1 apresenta um *template* que pode ser usado para geração de classes inicias de uma aplicação. Pode-se ver expressões e variáveis do gerador na cor alaranjada, e coloração diferente para a linguagem PHP como comentários na cor verde, variáveis na cor vermelha, palavras reservadas e funções na cor azul e *strings*[6] em vermelho claro.

6 Variável que armazena uma cadeia de caracteres.

```
...
1       /**
2       * @author  __AUTOR__
3       * @since   __DATE_F__
4       * @version       1.0
5       * @name    __TABELA__
6       * @abstract      classe responsável pelo tratamento de dados da tabela __TABELA__
7       */
8       class __TABELA__
9       {
10                  var $conn;//PDO
11                  var $last_insert;
12                  __I_PARA_TODOS_OS_ATRIBUTOS__
13                  /**
14                  * @var private __TIPO_DO_ATRIBUTO__
15                  * @abstract equivalente ao campo __ATRIBUTO__ da tabela __TABELA__
16                  */
17                  private $__ATRIBUTO__=false;
18                  __I_SEPARADOR__ \n\t\t\t __F_SEPARADOR__
19                  __F_PARA_TODOS_OS_ATRIBUTOS__
20      }
...
```

Quadro 1: Declaração inicial de um *template* para gerar uma classe simples em PHP

No Quadro 1 podem ser encontrados algumas das expressões e variáveis básicas do sistema, são elas:

- __AUTOR__ – Esta variável possui como valor o nome do autor do *template.* Esta variável é definida na inicialização do gerador de código.
- __DATE_F__ – Esta variável guarda a data final de execução do gerador, ou seja a data relativa ao termino de processamento do *template* pelo gerador.
- __TABELA__ – O valor desta variável varia para cada tabela. Ela guarda o nome da tabela que está sendo processada pelo gerador.
- __ATRIBUTO__ – Similar ao item anterior, o valor desta variável varia para cada campo da tabela e guarda o nome do campo da tabela que está sendo processado pelo gerador.
- __TIPO_DO_ATRIBUTO__ – Esta variável guarda o tipo do campo da tabela que está sendo processado pelo gerador.
- __I_PARA_TODOS_OS_ATRIBUTOS__ – Inicia uma expressão de replicativa.
- __I_SEPARADOR__ – Determina o inicio de um separador de códigos gerados. Este separador é opcional. O texto delimitado entre

separadores será colocado entre os códigos gerados dentro de uma expressão replicativa.

- __F_SEPARADOR__ – Determina o final de um delimitador.
- __F_PARA_TODOS_OS_ATRIBUTOS__ – Delimita o final de uma expressão replicativa.

O código entre as linhas 12 e 19 apresentado no Quadro 1 serão repetido para todos os campos e separados por uma nova linha e 3 tabulações.

No Quadro 2 surgem mais três identificadores do gerador:

- __I_DO_TIPO__ – Este identificador delimita para que tipo de campo campos os códigos serão replicados. Sem este identificador fica definido que os códigos serão gerados para todos os campos de uma tabela. PRI significa que os códigos serão gerados apenas para campos que são chaves primárias de uma tabela. Uma expressão replicativa com este termo pode ser considerada replicativa-condicional, só replicará códigos para campos que atenderem a condição.
- __F_DO_TIPO__ – Determina o final de uma condição de geração de código.
- __BD__ – Esta variável assume o valor do banco de dados trabalhado.

Os Quadros 1 e 2 mostram como desenvolver *templates* básicos para o gerador de código. Esta técnica faz o A partir da montagem de *templates* como os descritos nos Quadros 1 e 2 é possível criar e montar *templates* completos para um banco de dados inteiro.

```
...
1       /**
2       * @name __TABELA__
3       * @abstract Construtor da classe __TABELA__
4       * @author __AUTOR__
5       __I_PARA_TODOS_OS_ATRIBUTOS__
6       __I_DO_TIPO__ PRI __F_DO_TIPO__
7       __I_SEPARADOR__ \n __F_SEPARADOR__
8       * @param __TIPO_DO_ATRIBUTO__ $__ATRIBUTO__
9        __F_PARA_TODOS_OS_ATRIBUTOS__
10      */
11      function __TABELA__ //Construtor PHP4 para PHP5 coloque __contruct no lugar de __TABELA__
12      (
13       __I_PARA_TODOS_OS_ATRIBUTOS__
14       __I_DO_TIPO__ PRI __F_DO_TIPO__
15       __I_SEPARADOR__, __F_SEPARADOR__
16       $__ATRIBUTO__=false
17      __F_PARA_TODOS_OS_ATRIBUTOS__
18      )
19      {
20              if
21              (
22                __I_PARA_TODOS_OS_ATRIBUTOS__
23                __I_DO_TIPO__ PRI __F_DO_TIPO__
24                __I_SEPARADOR__ && __F_SEPARADOR__
25               $__ATRIBUTO__
26                __F_PARA_TODOS_OS_ATRIBUTOS__
27              )
28              {
29                      $this->busca__TABELA__por_campo_chave
30                      (
31                          __I_PARA_TODOS_OS_ATRIBUTOS__
32                          __I_DO_TIPO__ PRI __F_DO_TIPO__
33                          __I_SEPARADOR__ , __F_SEPARADOR__
34                          $__ATRIBUTO__
35                          __F_PARA_TODOS_OS_ATRIBUTOS__
36                       );
37
38              }
39              $this->conn = new PDO( 'mysql:host=localhost;dbname=__BD__' , 'usuario' , 'senha' );
40      }
...
```

Quadro 2: *Template* com uma função construtora.

```
...
1       /**
2       * @name __set
3       * @abstract altera o valor de um atributo da classe __TABELA__
4       * @author __AUTOR__
5       * @param $p
6       * @param $v
7       */
8       function __set($p,$v)
9       {
10              __I_PARA_TODOS_OS_ATRIBUTOS__
11              __I_SEPARADOR__ \n\t\t__F_SEPARADOR__
12              __I_DO_TIPO__ MUL __F_DO_TIPO__
13              if($p=="__ATRIBUTO__" && is_object($v))
14              {
15                      $v=$v->__ATRIBUTO_ESTRANGEIRO__;
16              }
17              __F_PARA_TODOS_OS_ATRIBUTOS__
18              $this->$p=$v;
19      }
...
```

Quadro 3: *Template* com um método magico.

O Quadro 3 mostra um *template* de um método mágico para alterar valores da a classe apresentada no Quadro 1, Neste *template* é dada uma atenção para campos que referenciam campos de outras tabelas, as chaves estrangeiras.

Neste quadro fica visível o uso de uma expressão repetitiva de códigos para campos que são chaves estrangeiras (através da expressão condicional da linha 12), para tratar atributos estrangeiros surge mais um identificador, o "__ATRIBUTO_ESTRANGEIRO__", seu uso é restrito em chaves estrangeiras e trás o valor do campo que o campo referencia da outra tabela.

O código do Quadro 3 intermediará o acesso externo aos atributos privados da classe e caso sejam passados objetos, o mesmo tentará tratá-los e extrair as informações úteis do objeto. Em analogia pode-se usar o identificador "__TABELA_DO_ATRIBUTO_ESTRANGEIRO__" para referenciar a tabela a qual o campo faz referencia.

3.4 CÓDIGOS RESULTANTES

A função de um *template* neste trabalho é ser usado pelo gerador para replicar um determinado código para as tabelas de um banco de dados MySQL de acordo com o código contido dentro do mesmo. Desta forma tem-se nos quadros seguintes os códigos-fonte resultantes para o banco de dados adotado como caso de uso.

Os Quadros 4 e 5 mostram como gerar de código a partir do *template* do Quadro 1 para as tabelas do Caso de uso. A padronização do código está clara em relação ao *template*, pode-se ver também que o código da linha 12 a linha 19 do Quadro 1 é replicado para todos os campos das tabelas do caso de uso.

```
...
/**
 * @author        Leonardo Weslei Diniz <leonardoweslei@gmail.com>
 * @since         07/12/2010
 * @version     1.0
 * @name         autores
 * @abstract     classe responsável pelo tratamento de dados da tabela autores
 */
class autores
{
    var $conn;//PDO
    var $last_insert;
    /**
    * @var private int(5)
    * @abstract equivalente ao campo codigo da tabela autores
    */
    private $codigo=false;
    /**
    * @var private varchar(150)
    * @abstract equivalente ao campo nome da tabela autores
    */
    private $nome=false;
    /**
    * @var private varchar(200)
    * @abstract equivalente ao campo email da tabela autores
    */
    private $email=false;
}
...
```

Quadro 4: Código gerado para a tabela autores a partir do primeiro *template definido no Quadro 1.*

```
...
/**
 * @author        Leonardo Weslei Diniz <leonardoweslei@gmail.com>
 * @since         07/12/2010
 * @version     1.0
 * @name         postagem
 * @abstract     classe responsável pelo tratamento de dados da tabela postagem
 */
class postagem
{
    var $conn;//PDO
    var $last_insert;
    /**
    * @var private int(10) unsigned zerofill
    * @abstract equivalente ao campo codigo da tabela postagem
    */
    private $codigo=false;
    /**
    * @var private text
    * @abstract equivalente ao campo texto da tabela postagem
    */
    private $texto=false;
    /**
    * @var private int(5)
    * @abstract equivalente ao campo dono da tabela postagem
    */
    private $dono=false;
    /**
    * @var private enum('POSTADO','SALVO')
    * @abstract equivalente ao campo situacao da tabela postagem
    */
    private $situacao=false;
}
...
```

Quadro 5: Código gerado para a tabela postagem a partir do primeiro *template definido no Quadro 1*.

Os Códigos dos Quadros 6, 7, 8 e 9 representam o resultado do *scaffolding* para os *templates* apresentados nos Quadros 2 e 3.

```
...
1   /**
2   * @name postagem
3   * @abstract Construtor da classe postagem
4   * @author Leonardo Weslei Diniz <leonardoweslei@gmail.com>
5   * @param int(10) unsigned zerofill $codigo
6   */
7   function postagem($codigo=false) //Construtor PHP4 para PHP5 coloque __contruct no lugar de postagem
8   {
9       if ($codigo)
10      {
11          $this->busca_postagem_por_campo_chave($codigo);
12      }
13      $this->conn = new PDO ( 'mysql:host=localhost;dbname=bd_exemplo' , 'root' , '123' );
14  }
...
```

Quadro 6: Código gerado para a tabela postagem a partir do segundo *template* definido no Quadro 2.

Nos códigos nota-se a substituição de identificadores de variáveis do *template* por metadados como o nome da tabela, do campo ou tipo do campo.

```
...
1   /**
2   * @name autores
3   * @abstract Construtor da classe autores
4   * @author Leonardo Weslei Diniz <leonardoweslei@gmail.com>
5   * @param int(5) $codigo
6   */
7   function autores($codigo=false) //Construtor PHP4 para PHP5 coloque __contruct no lugar de autores
8   {
9       if ($codigo)
10      {
11          $this->busca_autores_por_campo_chave($codigo);
12      }
13      $this->conn = new PDO ( 'mysql:host=localhost;dbname=bd_exemplo' , 'root' , '123' );
14  }
...
```

Quadro 7: Código gerado para a tabela autores a partir do segundo *template* definido no Quadro 2.

```
...
1     /**
2     * @name __set
3     * @abstract altera o valor de um atributo da classe postagem
4     * @author Leonardo Weslei Diniz <leonardoweslei@gmail.com>
5     * @param $p
6     * @param $v
7     */
8     function __set($p,$v)
9     {
10        if($p=="dono" && is_object($v))
11        {
12            $v=$v->codigo;
13        }
14        $this->$p=$v;
15    }
...
```

Quadro 8: Código gerado para a tabela postagem de acordo com terceiro *template* definido no Quadro 3.

```
...
1     /**
2     * @name __set
3     * @abstract altera o valor de um atributo da classe autores
4     * @author Leonardo Weslei Diniz <leonardoweslei@gmail.com>
5     * @param $p
6     * @param $v
7     */
8     function __set($p,$v)
9     {
10
11        $this->$p=$v;
12    }
...
```

Quadro 9: Código gerado para a tabela autores de acordo com o terceiro *template* definido no Quadro 3.

A diferença entre os códigos dos Quadros 8 e 9 vai além do nome dos campos e da tabela, o campo dono da da tabela postagem é na verdade uma referência ao campo codigo da tabela autores. Conforme comentado no Quadro 3, neste caso ao se alterar o valor do atributo dono da classe postagem, será feito o teste para verificar se o mesmo é um objeto criado a partir da classe autores, caso o resultado seja verdadeiro o valor do atributo dono passa então a ter o mesmo valor que o atributo codigo da classe autores.

4 CONCLUSÃO

De acordo com os resultados da geração de códigos percebe-se que ferramenta é útil, o resultado do caso de uso apresenta uma estrutura minimizada, mas essa a rotina e similar a adotada a de desenvolvimento de projetos que os desenvolvedores de sistemas seguem, e em uma escala maior o tempo gasto com estas tarefas gera custos que podem ser reduzidos com o uso do gerador de código.

Nos exemplos do casos de uso foi usado uma abstração de orientação a objeto, mas a ferramenta possibilita que o desenvolvedor faça o *template* da maneira que preferir, ou seja, se o mesmo necessitar de um *template* com código estruturado ou mesmo trabalhar com *frameworks* e precisar agilizar o trabalho de uma forma que o *scaffoldig* do *framework* não atenda, ele pode usar a ideia.

Como foram usados *templates* os códigos-fonte resultantes obedecem a um padrão definido pelo *template* com isso a manutenção posterior deles se torna fácil, pois a diferença esta nos dados usados.

Como o gerador funciona baseado em um banco de dados, desenvolver uma aplicação CRUD se torna uma tarefa simples, rápida e funcional. Com o gerador o desenvolvedor ganha mais tempo para garantir a qualidade da aplicação.

O gerador pode ser classificado como uma ferramenta CASE de apoio ao desenvolvedor pois automatiza as rotinas de codificação para o mesmo. Os código gerados pela ferramenta não ficam dependentes da mesma, com isso o desenvolvedor pode usá-la quando quiser ou quando achar necessário possibilitando aos códigos independência em relação ao gerador, padronização ao *template* e ganhos com relação ao desenvolvimento.

5 TRABALHOS FUTUROS

Para trabalhos futuros sugere-se que o mecanismo de apresentação e processamento do *template* seja melhorado, retirando algumas limitações nos mesmos. O *template* poderá apresentar mais possibilidades ao desenvolvedor para que ele não precise trabalhar nos códigos-fonte gerados mas somente nos *templates* relacionados a eles.

O gerador se encontra limitado ao SGBD MySQL, apesar do conceito ser parecido para outros SGBDs ainda não está implementado, é sugerido então que implemente-se a coleta e padronização de metadados para outros SGBDs.

Sugere-se também que o gerador também apresente um modelo de identificadores personalizável para dar ainda mais liberdade ao desenvolvedor, com isso o gerador seria totalmente adaptável ao desenvolvedor.

6 REFERÊNCIAS

ALLEN, Rob; LO, Nick; BROWN, Steven. **Zend Framework em Ação.** Editora Alta Books, 2009.

ALMEIDA, Allbert Velleniche de Aquino. **Histórico das linguagens**. Disponível em: <http://www.allbert.com.br/alg/Aula01-Introducao.pdf>. Acesso em: 15 de out. de 2010.

ALMEIDA, Israel Bastos de Souza; FRANÇA, ivo chaves de. **Ferramenta de captura de comandos DML em aplicações web com PHP e MySQL.** Salvador – BA: UNIVERSIDADE CATÓLICA DO SALVADOR, 2007.

AMBLER, S. **Modelagem Ágil: Praticas eficazes para a programação eXtrema e o Processo Unificado**. 1º Ed., Bookman, Porto Alegre, Brasil, 2004.

AZEVEDO, Manoel Santos. **Ferramenta CASE**, abr. [1998], Disponível em : <http://www.internext.com.br/mssa>. Acesso em: 27 mar. 2001.

ARNOLD, Pauline; WHITE, Percival. **A era da automação**. Rio de Janeiro: Lidador, 1965.

AVELINO, Izabel; KUWATA, Jefferson; BARRÉRE, Eduardo. **Construção de sites para comunidades virtuais e intranet utilizando CMS**. Disponível em: <http://www.aedb.br/seget/artigos06/602_Artigo_Construcao_de_Sites_com_CMS.pdf>. Acesso em 18 de nov. de 2010.

BARTIÉ, Alexandre. **Garantia da qualidade de software: adquirindo maturidade organizacional.** Rio de Janeiro: Editora Campus, 2002.

BROWN, Greg. **Uma Introdução aos frameworks**. Disponível em: <http://www.revistaphp.com.br/artigo.php?id=210>. Acessado em 28 de out. de 2010.

CAKEPHP. Disponível em: <http://cakephp.org/>. Acessado em 10 de out. de 2010.

CALIARI, Fábio M. **Ferramentas CASE**. São Bento do Sul: Universidade do Estado de Santa Catarina. Disponível em: <http://www.joinville.udesc.br/sbs/professores/caliari/materiais/Aula_05___Ferramenta_Case.pdf>. Acesso em: 29 de set. de 2010.

CodeIgniter. Disponível em: <http://www.codeigniter.com/>. Acessado em 10 de set. de 2010.

DEMPSEY, L. and HEERY, R. **Metada: A Current View of Practice and Issues**. Journal of Documentation, v. 54, n.2, mar. 1998.

FERRAZ, Ronaldo M. **Rails para sua Diversão e Lucro.** Disponível em: <http://kb.reflectivesurface.com/br/tutoriais/railsDiversaoLucro/tutorial.odt>. Acesso em 18 de out. de 2010.

FERRAZ, Ronaldo Melo. **Evolução das Linguagens de Programação.** Disponível em: <http://www.doctum.com.br/hebert/admin/arquivos/files/624b7d3d02e4c1389948d8ad821bd7e8.pdf>. Acesso em 21 de out. de 2010.

FONTANA, Marines. **Avaliação de desempenho de sistemas gerenciadores de banco de dados.** Nova Hamburgo, 2006. Disponível em: <http://tconline.feevale.br/tc/files/737.doc>. Acesso em: 19 de set. de 2010.

GABARDO, Ademir Cristiano. **CodeIgniter Framework PHP.** Editora Novatec, 2010.

HOFFMANN, Tatiana M. **Avaliação da qualidade da ferramenta CASE *system architect* baseada na norma ISO/IEC 14102** BLUMENAU-SC: UNIVERSIDADE REGIONAL DE BLUMENAU,2001. Disponível em: <http://campeche.inf.furb.br/tccs/2001-I/2001-1tatianamielehoffmannvf.pdf>. Acesso em 22 de out. de 2010.

HOWE, D. **Free on-line Dictionary of Computing (FOLDOC),** 1996. Disponível em: <http://wombat.doc.ic.ac.uk/>. Acesso em 22 de set. de 2010.

IBARRA, Gustavo B.; Willemann,David P. **Framework de apoio ao desenvolvimento de aplicações *web* com banco de dados, utilizando *Struts, Tiles* e *Hibernate***. Florianópolis-SC: UNIVERSIDADE FEDERAL DE SANTA CATARINA, 2007.

KLUG, M. **Gerador de código JSP Baseado em Projeto de Banco de Dados**. 2007. Trabalho de Conclusão de Curso de Ciência da Computação, UNIVERSIDADE REGIONAL DE BLUMENAU, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2007-I/2007-1maiconklugvf.pdf>. Acesso em: 28 de out. de 2010.

LEAL, Marcelio D'Oliveira. **ClassGenerator: um gerador de artefatos multiplataforma.** Belém, 2005. Disponível em: <http://marcelioleal.files.wordpress.com/2008/08/tccfinal.pdf>. Acesso em: 23 de out. de 2010.

LEITE, Julio Cesar Sampaio do Prado. **Gerenciando a Qualidade de Software com Base em Requisitos.** Rio de janeiro: PUC RJ. Disponível em: <http://www-di.inf.puc-rio.br/~julio/Slct-pub/Livro-qualidade.pdf>. Acessado em 29 de out. de 2010.

LOH, Stanley. Ambiente de **desenvolvimento de software e ferramenta CASE**, mar. 1996, Disponível em : <http://atlas.ucpel.tche.br/~loh>. Acesso em: 27 mar. 2001.

MASETO, Jhony Maiki**. UMA ANÁLISE ENTRE FRAMEWORKS DE PHP**. Chapecó-SC: UNIVERSIDADE COMUNITÁRIA REGIONAL DE CHAPECÓ, 2006.

MILANI, André. **MySQL - Guia do Programador.** Editora Novatec, 2007.

MINETTO, Elton Luis. **Frameworks para Desenvolvimento em PHP**. São

Paulo: Novatec, 2007.

NIEDERAUER, J. **PHP com XML: guia de consulta rápida**. São Paulo: Novatec, 2002.

OLIVEIRA JR., Eustáquio Rangel de. **Ruby - Conhecendo a Linguagem**. BRASPORT, 2006.

PERES, Sarajane Marques. **Ferramentas CASE**. Disponível em: <http://www.din.uem.br/~mmdias/Ambientes/FerramentasCASE_Sarajane.ppt>. Acesso em 22 de set. de 2010.

Manual do PHP. Disponível em: <http://www.php.net/manual/pt_BR/index.php>. Acessado em 10 de set. de 2010.

PORTELLA, Cristiano R. R.. **Conceitos de Qualidade em Software, Campinas**: PUC Campinas, 2006. Disponível em: <http://www.cesarkallas.net/arquivos/faculdade/engenharia_de_software/17-Qualidade%20em%20Software/Conceitos%20de%20Qualidade%20em%20SW.pdf>. Acessado em 28 de out. de 2010.

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

REIS, Karen. **Comércio eletrônico sistemas para internet.** Faculdades Metropolitanas Unidas. Disponível em: <http://www.karenreis.com.br/fmu/CE_SistemasParaInternet_Aula06.pdf>. Acesso em 26 de out. de 2010.

RIVELO, Cassio Luiz. **Um Sistema Web para Acompanhamento de Protocolo.** Faculdade de Engenharia de Guaratinguetá. Disponível em: <http://www.feg.unesp.br/ceie/Monografias-Texto/CEIE0401.pdf>. Acesso em: 27 de out. de 2010.

ROBBINS, Stephen P. **Comportamento Organizacional**. 8 ed. Rio de

Janeiro: LTC, 1999.

ROCHA, Allex Motta Melo da. **Introdução a Arquitetura de Software.** Belém-PA: Fábrica de Software do CESUPA - Centro Universitário do Pará, 2010. Disponível em: <http://www.fabsoft.cesupa.br/site/images/introducao_arquitetura_software.pdf>. Acesso em 03 de nov. de 2010

RUMBAUGH, J.; BLAHA ,Michael; PREMERLANI ,William,; EDDY ,Frederick; LORENSEN ,William. **Modelagem e projetos baseados em objetos**. Tradução Dalton Conde de Alencar. Rio de Janeiro: Campus, 1994.

SANTOS, Lincoln Fernandes Paulino dos; AMARAL, Aline Maria Malachini Miotto. **Construção de um framework para o desenvolvimento de aplicações web.** Encontro Internacional de Produção Científica Cesumar de 27 a 30 de outubro de 2009. Centro Universitário de Maringá. Disponível em: <http://www.cesumar.br/epcc2009/anais/lincoln_fernandes_paulino_santos.pdf>. Acesso em: 11 de nov. de 2010.

SAUVÉ, Jacques. **Projeto de software orientado a objeto programa**. Universidade Federal Campinas Grande. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 21 de out. 2010.

SEMELER, Alexandre Ribas. **Método para elaboração de objetos de aprendizagem**. Porto Alegre, 2006. Disponível em: http://www6.ufrgs.br/neiti/Artigos/Alexandre.pdf. Acesso em: 27 de out. de 2010.

SILVA, Enio Kilder Oliveira da. **Um estudo sobre sistemas de banco de dados cliente/servidor.** Associação Paraibana de Ensino Renovado. João Pessoa – PB. Disponível em: <http://www.biblioteca.sebrae.com.br/bds/BDS.nsf/66D405293751CC5B03256D520059B6F6/$File/190_1_arquivo_bdados.pdf>. Acesso em: 01 de nov. de 2010.

SILVA, Geiza Maria Hamazaki da. **Extração de Conteúdo Computacional de Provas Intuicionistas.** Pontifícia Universidade Católica do Rio de Janeiro. isponível em: <http://www2.dbd.pucrio.br/pergamum/tesesabertas/9924916_04_pretextual.pdf>. Acesso em: 22 de out. de 2010.

SOUZA, Cleidson R. B. **Interfaces de Programação de Aplicações**. Universidade Federal do Pará. Disponível em: <http://www2.ufpa.br/cdesouza/teaching/labes/apis.pdf>. Acesso em: 21 de out. de 2010.

SOUZA, Marcos Vinícius Bittencourt de. **Estudo Comparativo entre Frameworks Java para Construção de Aplicações Web**. Santa Maria - RS: Universidade Federal de Santa Maria, 2004.

VILARINHO, Louriel Oliveira. **Automatização e automação.** Universidade Federal de Uberlândia. Uberlândia – GO. Disponível em: <http://www.mecanica.ufu.br/old/ArquivosDisciplinas/DEM27_0175.PDF>. Acessado em 28 de out. de 2010.

WTHREEX. **Conceitos: Arquitetura de Software.** Disponível em: <http://www.wthreex.com/rup/process/workflow/ana_desi/co_swarch.htm#A %20Typical%20Set%20of%20Architectural%20Views>. Acessado em: 29 de nov. de 2010.

ZEND FRAMEWORK. Disponível em: <http://framework.zend.com/>. Acessado em 10 de set. de 2010.