

**CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA DE JAHU
CURSO DE TECNOLOGIA EM GESTÃO DA TECNOLOGIA DA
INFORMAÇÃO**

JEFERSON FERNANDO MATOSO

MATHEUS FURTADO

**IMPACTO DO USO DE DESIGN PATTERNS NO DESENVOLVIMENTO DE
SOFTWARE**

JAHU, SP

2º semestre/2015

JEFERSON FERNANDO MATOSO

MATHEUS FURTADO

**IMPACTO DO USO DO DESIGN PATTERNS NO DESENVOLVIMENTO DE
SOFTWARE**

Monografia apresentado como exigência para conclusão do curso de Graduação em Gestão da Tecnologia da Informação pela Faculdade de Tecnologia de Jahu – FATEC – JAHU, sob orientação do Prof. Icaro Saggioro.

JAHU, SP

2º semestre/2015

AGRADECIMENTOS

Agradecemos primeiramente a Deus que se faz muito importantes em nossas vidas sendo indispensável em qualquer momento.

Agradecemos ao professor orientador do projeto, Icaro Saggioro e aos professores Sérgio Alexandre de Castro e Adriana Bertoldi Carreto de Castro que contribuíram diretamente e indiretamente no desenvolvimento deste trabalho.

Não poderíamos deixar de agradecer nossos familiares e amigos, que deram apoio durante todo o curso.

RESUMO

Este Trabalho de Conclusão de Curso aborda o impacto da implementação de Padrões de Projeto em desenvolvimento de software. Apresentará definições conceituais teóricas sobre Design Patterns nas áreas de Engenharia de Software e Arquitetura de Software. Após abordarmos o assunto com base em estudos teóricos, realizaremos análise do impacto da utilização de padrões de projetos em uma empresa de desenvolvimento de software, definindo e detalhando as técnicas utilizadas, as dificuldades enfrentadas e os objetivos esperados.

Palavras-Chave: Padrões de Projetos; Arquitetura de Software; Engenharia de Software; Desenvolvimento de Software.

ABSTRACT

This paper presents on approach to the impacts of Design Patterns implementation on software development. It will be described conceptual aspects on Design Patterns about Software Engineering Architecture. After a theoretic aspect approach it is done an analysis of the impact of using design patterns in a software development company, describing the techniques, the difficulties, and the expected objectives.

Keywords: Design Patterns, Software Engineering, Software Architecture, Software Development

LISTA DE ILUSTRAÇÕES

Figura 1	19
Figura 2	22
Figura 3	26
Figura 4	27
Figura 5	28
Figura 6	29
Figura 7	34
Figura 8	35
Figura 9	59
Figura 10	60
Figura 11	61

LISTA DE TABELAS

Tabela 1	21
----------------	----

SUMÁRIO

1.	INTRODUÇÃO	10
1.1	Objetivo	10
1.2	Metodologia	11
1.3	Estrutura do Trabalho	11
2.	PADRÕES DE PROJETO	13
2.1	História dos Padrões de Projeto.....	13
2.2	Padrões de Projeto na Engenharia de Software	15
2.3	Características de Padrões de Projetos de Software	17
2.4	Tipos de Padrões de Projeto em Software.....	19
2.5	Considerações Finais	23
3.	ARQUITETURA DE SOFTWARE.....	24
3.1	A Arquitetura de Software	24
3.2	Padrões de Projeto na Arquitetura de Software.....	25
3.3	MVC (Model-View-Controller)	25
3.4	DAO (Data Access Object)	28
3.5	Business Object (regras de negócio)	29
3.6	Considerações Finais	30
4	APLICAÇÃO DE DESIGN PATTERNS.....	31
4.1	Considerações Iniciais	31
4.2	Sobre a empresa.....	31
4.3	Portfólio de soluções.....	32
4.4	A busca por sustentabilidade	32
4.5	Evidências sobre o projeto	35
4.5.1	Singleton - Garantia de unicidade simplifica o controle	36
4.5.2	DAO - Estrutura central para persistência multibanco	39
4.5.3	Façade – Simplificação de uma biblioteca de código.....	43
4.5.4	MVC – Divisão de tarefas e responsabilidade	46
4.5.5	FactoryMethod – Criação de objetos simplificada.....	50
4.6	Resultados esperados	51
4.7	Desenvolvimento do Projeto.....	52
5	CONCLUSÃO	54
	BIBLIOGRAFIA	56
	ANEXO A.....	57

Os módulos do WMS são divididos em:.....	58
1. Automação	58
2. Visualização do depósito	58
3. Integrações.....	58
4. Business Intelligence	58
5. Consultas Web.....	59
6. Gestão de entregas	59
7. Faturamento de serviços.....	59
8. Emissão de livros fiscais	59
9. Nota fiscal eletrônica.....	59
ANEXO B	62

1. INTRODUÇÃO

O presente trabalho foi desenvolvido com o intuito de ser integrado ao projeto “Análise das possibilidades de Desenvolvimento de Startups e Empresas de Tecnologia da Informação na Cidade de Jaú e Região”, desenvolvido pela professora Adriana Bertoldi Carretto de Castro. O projeto de pesquisa tem como objetivo efetuar um estudo de caso junto às empresas de tecnologia da informação de Jaú e região e inserir os alunos da faculdade neste processo. Em reunião realizada com os empresários de TI, de Jaú e região, em maio de 2014, foi sugerido pelos empresários que os alunos desenvolvessem projetos que contribuíssem para a integração entre a faculdade (mais especificamente o curso de Gestão da Tecnologia da Informação) e as empresas. Assim, através de contato com as empresas, foram levantados problemas enfrentados pelas empresas, e foi sugerido aos alunos que desenvolvessem soluções para os problemas.

Neste trabalho de conclusão de curso de graduação em Gestão da Tecnologia da Informação tem por objetivo o estudo do impacto da utilização de Design Patterns no desenvolvimento de software.

Durante o decorrer do trabalho será apresentado os estudos realizados sobre Design Patterns (Padrões de Projetos), desde a sua origem como padrões de projeto na construção civil, até a adoção da metodologia em projetos de softwares.

Foi utilizado um estudo de caso com a aplicação em ambiente profissional dos Padrões de Projeto no desenvolvimento de software de um novo projeto em uma empresa de desenvolvimento de software ativa no mercado.

1.1 Objetivo

Este trabalho busca identificar os benefícios, dificuldades, facilidades e malefícios que a utilização dos padrões trazem em um projeto de software, utilizando por base os estudos em bibliografia sobre o tema e com os resultados obtidos no estudo de caso. Na qual haverá comparação com o teórico com o ambiente prático.

Este trabalho tem por objetivo responder a seguinte pergunta: Qual o peso da utilização de Padrões de Projeto no desenvolvimento de sistemas de automação? E com esta questão em mente vamos identificar os impactos, positivos e negativos

que a implementação de Design Patterns causa a todos os envolvidos no novo projeto.

1.2 Metodologia

Para o desenvolvimento deste trabalho foi utilizado a metodologia de pesquisa-ação, no qual realizamos consultas em livros sobre o tema de padrões de projetos, no qual abordavam a sua origem, padrões de projeto voltados para o desenvolvimento de software, arquitetura de software e engenharia de software.

Também foi utilizado de estudos realizados por profissionais conceituados da área de desenvolvimento de software, no qual os mesmos compartilham de suas experiências na utilização dos padrões de projetos no desenvolvimento de software.

Para exemplificar todo o conteúdo obtida através dos estudos de referências bibliográficas e os conhecimentos teóricos obtidos durante todo o trabalho, foram aplicados os conceitos estudados no desenvolvimento de software, que observamos o impacto que a utilização do padrão de projetos causam em um novo projeto de desenvolvimento.

1.3 Estrutura do Trabalho

Durante os próximos capítulos serão apresentados os conteúdos obtidos através dos estudos teóricos e práticos realizados para o desenvolvimento deste trabalho.

Os capítulos estão organizados da seguinte forma: no capítulo “Padrões de Projeto” será apresentado a origem dos padrões de projeto, apresentando o seu surgimento na execução de projetos de arquitetura de construção civil, e como foi o início de sua utilização no desenvolvimento de software. Também serão abordados os principais padrões utilizados e o impacto que eles trouxeram para o desenvolvimento de softwares.

No capítulo “Arquitetura de Software” será abordado a utilização dos padrões de projetos voltados na área de arquitetura de software, tratado suas particularidades e impactos.

A apresentação dos resultados obtidos na aplicação prática de Design Patterns será descrito no capítulo “Aplicação de Design Patterns”, no qual é detalhado as etapas do projeto e os resultados obtidos com o estudo. Aqui será feito

uma comparação com os estudos teóricos obtidos, com o estudo em ambiente real de produção com a utilização da ferramenta de padrões de projeto.

2. PADRÕES DE PROJETO

Neste capítulo será descrito o surgimento, definições e os principais conceitos para os padrões de projetos na Engenharia de Software. Há descrito os principais padrões de projetos utilizados atualmente para o desenvolvimento de softwares.

2.1 História dos Padrões de Projeto

O conceito de Padrões de Projeto foi criado a partir da década de 70, através dos trabalhos desenvolvidos por Christopher Alexander, na qual associou o termo *Padrão* às repetições existentes em projetos de arquitetura, no qual se criou Padrões de Projetos voltados a projetos da construção civil.

Através de observações de construções da época, Alexander observou que construções distintas possuíam similaridade nas soluções dos mesmos problemas que eram enfrentados.

A partir de suas percepções na construção civil, Christopher Alexander em 1979 definiu conceitos e elaborou inúmeros padrões de projetos voltados para sua área de atuação, a arquitetura urbana.

Christopher Alexander, afirma no seu livro *“A Patterns Language”*. New York, NY (USA): Oxford University Press, 1977 que: “cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca o fazer da mesma maneira”.

Outro conceito colocado por Alexander em seu livro *“A Timeless Way of Building”*, 1979: “cada padrão é uma regra de três partes, que expressa uma relação entre um certo contexto, um problema e uma solução”. Sendo assim, segundo suas definições, para saber a real necessidade e viabilidade da existência de um padrão, é necessário dividir o padrão em três partes, sendo o problema, a solução e o contexto em que será aplicado.

Alexander estabelece que um padrão deve ter, idealmente, as seguintes características:

- **Encapsulamento:** um padrão encapsula um problema/solução bem definida. Ele deve ser independente, específico e formulado de maneira a ficar claro onde ele se aplica;

- **Generalidade:** todo padrão deve permitir a construção de outras realizações a partir deste padrão;

- **Equilíbrio:** quando um padrão é utilizado em uma aplicação, o equilíbrio dá a razão, relacionada com cada uma das restrições envolvidas, para cada passo do projeto. Uma análise racional que envolva uma abstração de dados empíricos, uma observação da aplicação de padrões em artefatos tradicionais, uma série convincente de exemplos e uma análise de soluções ruins ou fracassadas pode ser a forma de encontrar este equilíbrio;

- **Abstração:** os padrões representam abstrações da experiência empírica ou do conhecimento cotidiano;

- **Abertura:** um padrão deve permitir a sua extensão para níveis mais baixos de detalhe;

- **Combinatoriedade:** os padrões são relacionados hierarquicamente. Padrões de alto nível podem ser compostos ou relacionados com padrões que endereçam problemas de nível mais baixo.

Além da definição das características de um padrão, Alexander definiu o formato que a descrição de um padrão deve ter. Ele estabeleceu que um padrão deve ser descrito em cinco partes:

- **Nome:** uma descrição da solução, mais do que do problema ou do contexto;

- **Exemplo:** uma ou mais figuras, diagramas ou descrições que ilustrem um protótipo de aplicação;

- **Contexto:** a descrição das situações sob as quais o padrão se aplica;

- **Problema:** uma descrição das forças e restrições envolvidos e como elas interagem;

- **Solução:** relacionamentos estáticos e regras dinâmicas descrevendo como construir artefatos de acordo com o padrão, frequentemente citando variações e formas de ajustar a solução segundo as circunstâncias. Inclui referências a outras soluções e o relacionamento com outros padrões de nível mais baixo ou mais alto.

Com base nas definições e conceitos colocados por Christopher Alexander sobre padrões de projeto, Peter Jandl Jr DCA/Unicamp (2002), as características dos padrões de projeto são: “estes devem descrever e justificar soluções para problemas concretos e bem definidos”, “estes devem ser comprovados, isto é, devem ter sido previamente experimentados e testados” e ainda, “tratar problemas que ocorram em diferentes contextos”.

2.2 Padrões de Projeto na Engenharia de Software

Após os conceitos criados por Christopher Alexander, o surgimento de conceitos de padrões de projetos voltados especificamente para o desenvolvimento de software aconteceu no fim da década de 80 e durante toda a década de 90.

No livro “Engenharia de Software – Uma abordagem Profissional” de Roger S. Pressman, ele define projeto baseado em padrão como:

“Cria uma nova aplicação através da busca de um conjunto de soluções comprovadas para um conjunto de problemas claramente delineados. Cada Problema (e sua solução) é descrito por um padrão de projeto que foi catalogado e investigado por outros engenheiros de software que depararam com o problema e implementaram a solução ao projetarem outras aplicações. Cada padrão de projeto nos oferece uma abordagem comprovada para parte do problema a ser resolvido. ”

A primeira “aparição” de padrões de projetos voltados para desenvolvimento de software surgiu na conferência sobre programação orientada a objetos (OOPSLA) de 1987, durante o Workshop sobre Especificação e Projeto para Programação Orientada a Objetos (Beck 87). A partir de então, foram surgindo trabalhos abordando padrões de projetos em software, no qual são descritas soluções para problemas que ocorrem frequentemente no desenvolvimento de software e que podem ser reaproveitadas por outros desenvolvedores em projetos distintos.

Desde que se iniciou a utilização de padrões em projetos de desenvolvimento de software foram criados conceitos de padrões voltados para a área de desenvolvimento, uma das definições voltadas para software é a de Buschmann (1996): “Um padrão descreve uma solução para um problema que ocorre com

frequência durante o desenvolvimento de software, podendo ser considerado como um par problema/solução”.

Ainda, outra definição do conceito de “*Padrão*” nos projetos de desenvolvimento de software é de APPLETON (1997): “um padrão é um conjunto de informações instrutivas que possui um nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem-sucedidas para um problema repetido que ocorre sob determinado contexto e um conjunto de repercussões. ”

Um dos primeiros trabalhos estabelecendo padrões foi o de Coad, no qual são descritos sete padrões de análise. Em 2000, foi publicado o livro “Padrões de Projeto” por um grupo que ficou conhecidos como GoF (Gang of Four) por serem quatro escritores, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, no qual foram escritos vinte e três padrões de projetos, que servem de referência até os dias atuais.

Segundo Gamma (2000), projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que os redescobrir, mostrando que quando se faz uso de padrões de projetos, as soluções de problemas ocorrem mais facilmente.

Na tentativa de definir uma base sólida a partir da qual os padrões de projeto são identificados e se tornam úteis para as pessoas, Fowler (2006) descreve que os padrões de projeto podem ser identificados através da observação de problemas que ocorrem na prática, e que a descoberta e a definição de novos padrões estão relacionadas à experiência acerca de determinado problema. Partindo desse princípio, pode-se afirmar que a descoberta de novos¹³ padrões se dão através da observação de projetos que funcionam e da busca pela essência da sua solução, esse é um processo que exige grande capacidade de análise e experiência na abordagem do problema.

A definição de Fowler (2006) é muito semelhante a abordagem dada por Roger S. Pressman (1995), no qual Pressman diz que o ser humano evolui a partir da percepção de padrões durante toda sua existência:

Uma das razões para os engenheiros de software interessarem-se (e intrigados) em padrões de projeto é o fato de os seres humanos serem inerentemente bons no reconhecimento de padrões. Se não fossemos, teríamos parado no tempo e no espaço – incapazes de aprender de experiência passada, desinteressados em nos aventurarmos devida à nossa inabilidade de reconhecer situações que talvez nos levassem a correr altos riscos, transtornados por um mundo que parece não ter regularidade ou consistência lógica. Felizmente, nada disso acontece porque efetivamente reconhecemos padrões em praticamente todos os aspectos de nossas vidas.

2.3 Características de Padrões de Projetos de Software

A característica geral para padrões de projetos em desenvolvimento de software é que o padrão implementado deve solucionar problemas recorrentes em diferentes projetos. Além dessa característica primordial para um padrão, temos outras definições.

Coplien (2005) caracteriza um padrão de projeto eficaz da seguinte maneira:

- Ele soluciona um problema: Os padrões capturam soluções, não apenas estratégias ou princípios abstratos;
- Ele é um conceito comprovado: Os padrões apreendem soluções com um histórico, não teorias ou especulação;
- Uma solução não é óbvia: Muitas técnicas para resolução de problemas (como paradigmas ou métodos de projeto de software) tentam obter soluções com base nos primeiros princípios. Os melhores padrões geram uma solução para um problema diretamente – uma abordagem necessária para os problemas mais difíceis de projeto.

Gamma (2000) também define quatro elementos essenciais para um padrão:

1. O **nome** do padrão é uma referência que podemos usar para descrever um problema de projeto, suas soluções e consequências em uma ou duas palavras. Dar nome a um padrão aumenta imediatamente o nosso vocabulário de projeto. Isso nos permite projetar em um nível mais alto de abstração [...]

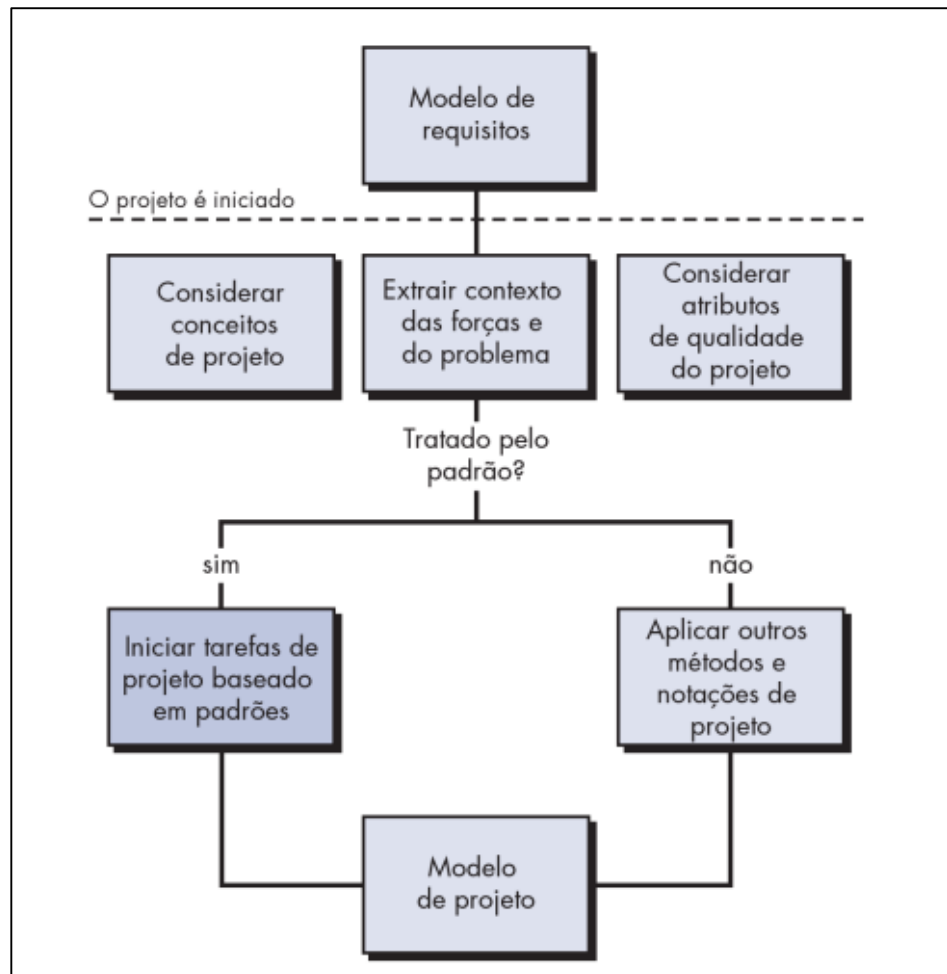
2. O **problema** mostra em que situação aplicar o padrão. Ele explica o problema e seu contexto. Pode descrever problemas de projeto específicos, tais como representar algoritmos como objetos. Pode descrever estruturas de classe ou objeto sintomáticas de um problema inflexível. Algumas vezes o problema incluirá uma lista de condições que devem ser satisfeitas para que faça sentido aplicar o padrão.

3. A **solução** descreve os elementos que compõe o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações. A solução não descreve um projeto concreto ou uma implementação em particular porque um padrão é um gabarito que pode ser aplicado em muitas situações diferentes. Em vez disso o padrão fornece uma descrição abstrata de um problema de projeto e de como um arranjo geral de elementos (classes e objetos) o resolve.

4. As **consequências** são os resultados e análises das vantagens e desvantagens (trade offs) da aplicação do padrão. Embora as consequências sejam raramente mencionadas quando descrevemos decisões de projeto, elas são críticas para a avaliação de alternativas de projetos e para a compreensão de custos e benefícios da aplicação do padrão. (GAMMA et al., 2000, p. 19, grifo do autor).

Uma característica muito importante destacada por Pressman (1995), é que um padrão será adotado no desenvolvimento de um software, após passar por uma análise de requisitos e contexto no qual o padrão será empregado. Pois a partir das definições do contexto do projeto, o projetista terá bem clara as forças que agiram sobre o projeto, assim identificando possíveis problemas, podendo encontrar soluções já disponíveis, assim aplicando uma abordagem de projeto baseado em padrões. A Figura 1 ilustra o modelo de contexto do projeto baseado em padrões.

Figura 1 – Contexto do projeto baseado em padrões



Fonte: Pressman (1995, p. 323)

2.4 Tipos de Padrões de Projeto em Software

Dentro de padrões de projetos há uma divisão dos padrões de acordo com suas finalidades. Essa classificação permite compreender os padrões mais rapidamente, e direcionar esforços na descoberta de novos padrões.

Pressman (1995) classifica os padrões em duas situações primeiramente, um é o padrão “generativo”, no qual descreve um problema, um contexto e forças, mas também descreve uma solução pragmática para o problema. E também há padrão “não generativo”, no qual ele descreve um contexto e um problema, mas não oferece nenhuma solução explícita.

Após esta classificação dada por Pressman (1995), existem os tipos categorias de padrões citadas por ele. São elas:

- Os padrões de projeto abrangem um amplo espectro de abstração e aplicação. Os padrões de arquitetura descrevem problemas de projeto de caráter amplo e diversos resolvidos usando-se uma abordagem estrutural;
- Os padrões de dados descrevem problemas orientados a dados recorrentes e as soluções de modelagem de dados que podem ser usadas para resolvê-los;
- Os padrões de componentes tratam de problemas associados ao desenvolvimento de subsistemas e componentes, a maneira através da qual eles se comunicam entre si e seu posicionamento em uma arquitetura maior;
- Os padrões de projeto de interfaces descrevem problemas comuns de interface do usuário e suas soluções com um sistema de forças que inclua características específicas de usuários finais;
- Os padrões para WebApp tratam de um conjunto de problemas encontrados ao se construir WebApps e em geral incorpora muitas das demais categorias de padrões;
- Idiomas descrevem como implementar todos ou parte de um algoritmo específico ou estrutura de dados para um componente de software no contexto de uma linguagem de programação específica.

Gamma et al. (2000) classifica os padrões de projeto a partir de dois critérios. O primeiro critério é finalidade, que reflete o que o padrão faz. Dentre o critério de finalidade, há padrões de criação, estrutural e comportamental. Segundo definição de Gamma et al. (2000):

Os padrões de criação se preocupam com o processo de criação de objetos. Os padrões estruturais lidam com a composição de classes ou de objetos. Os padrões comportamentais caracterizam as maneiras pelas quais classes ou objetos interagem e distribuem responsabilidades.

No segundo critério é o escopo, que especifica se o padrão se aplica primariamente a classe ou a objetos, em que os padrões para classe lidam com os relacionamentos entre classes e suas subclasses. Nos padrões para objetos lidam com relacionamento entre objetos que podem ser mudados em tempo de execução e são mais dinâmicos.

Representando essa classificação dos padrões de acordo com os critérios de finalidade e escopo, os padrões criados por Gamma (2000) pode ser organizado da forma representada no Quadro 1.

Tabela 1 – Classificação dos Padrões de Projeto

		Propósito		
		De Criação	Estrutural	Comportamental
Escopo	Classe	Factory Method	Adapter (Class)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (Obeject) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Momento Observer State Strategy Visitor

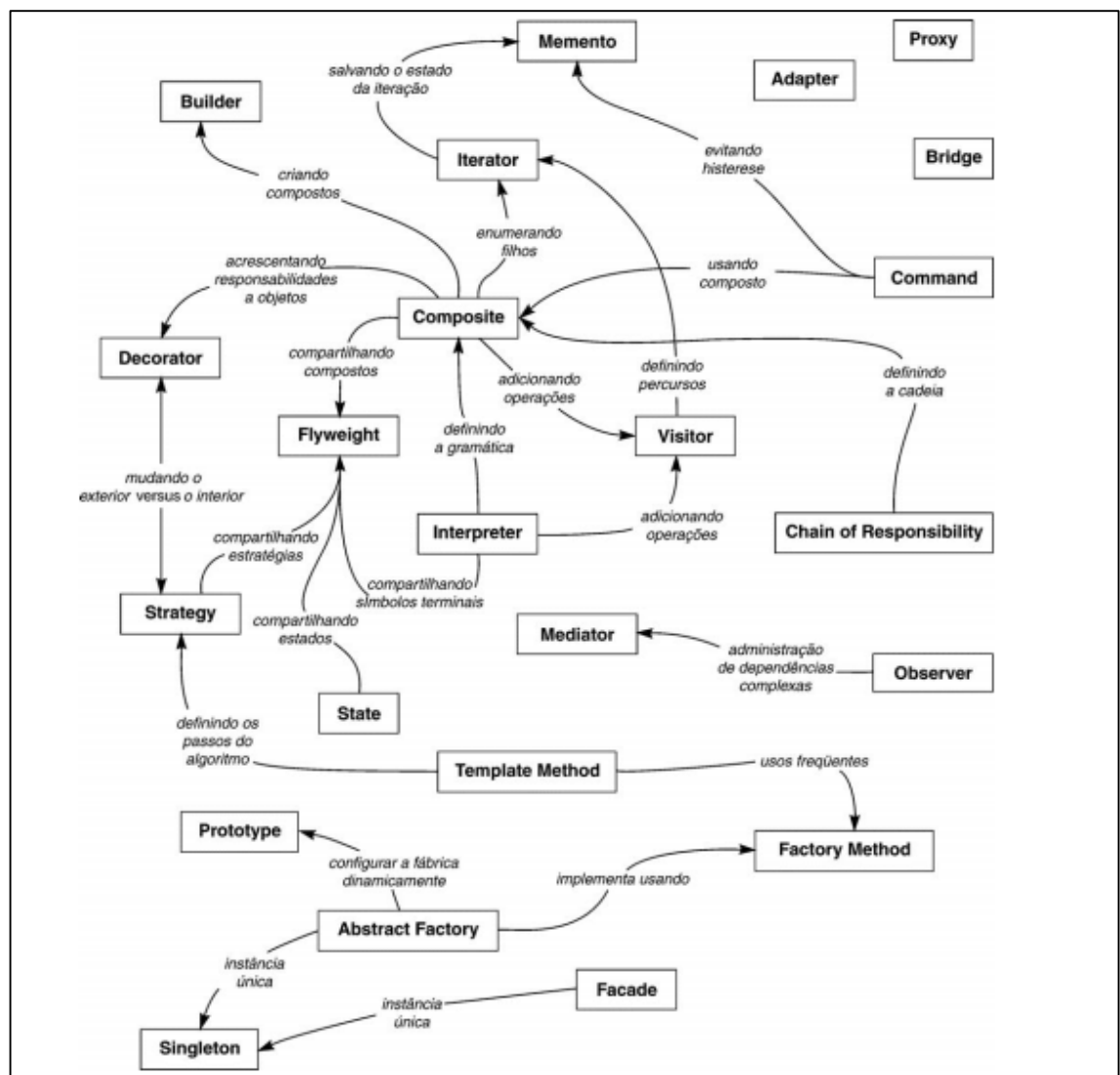
Fonte: Adaptado de Gamma et al. (2000, p. 26)

Em relação ao escopo, há dois conjuntos de padrões, voltado para classe e outro para objeto. Em relação ao propósito, existem três conjuntos de padrões com diferentes finalidades: de criação, estrutural e comportamental. Os padrões de criação no escopo de classe delegam o processo de criação de objetos para as subclasses, enquanto que os de escopo de objeto postergam esse processo para outros objetos. Os padrões estruturais no escopo de classe utilizam a herança para compor classes, enquanto que os de escopo de objeto descrevem estruturas para relacionar objetos. E os padrões comportamentais no escopo de classe utilizam a herança para descrever algoritmos e fluxos de controle, enquanto que aqueles

voltados para objetos descrevem como um grupo de objetos se relaciona e compartilha a execução de tarefas (GAMMA et al., 2000).

A Figura 2 ilustra graficamente as relações entre os padrões de projeto de acordo com o modo que os padrões mencionam seus inter-relacionamentos.

Figura 2 – Classificação dos Padrões de Projeto



Fonte: Gamma et al. (2000, p. 27)

2.5 Considerações Finais

Com os estudos realizados durante este capítulo, foi possível identificar o surgimento dos padrões de projetos e sua principal aplicação no desenvolvimento de software.

Também foi apresentado os principais padrões de projetos utilizados e seus benefícios. No próximo capítulo será abordado a utilização de padrões de projeto na área de arquitetura de software.

3. ARQUITETURA DE SOFTWARE

Neste capítulo será estudado os padrões de projeto na arquitetura de software, o que o uso da ferramenta de Design Patterns impacta na área da arquitetura de software.

Também será abordado o conceito básico de arquitetura de software, no qual será situado em que momento o padrão de projetos interage com a arquitetura de software.

3.1 A Arquitetura de Software

A arquitetura de software refere-se à organização geral do software e aos modos pelos quais disponibiliza integridade conceitual para um sistema (Shaw 1995). Pressman (1995) define arquitetura de software como estrutura ou a organização de componentes do programa (módulos), a maneira através da qual esses componentes interagem e a estrutura de dados são usadas pelos componentes. E com o uso de projetos de arquitetura de software permite a um engenheiro de software reusar soluções-padrão para problemas similares.

Shaw e Garlan (1995) definiriam um conjunto de propriedades que devem ser especificadas como parte de um projeto de arquitetura. São elas:

- **Propriedades estruturais:** Esse aspecto do projeto da representação da arquitetura define os componentes de um sistema (por exemplo, módulos, objetos, filtros) e a maneira pela qual os componentes são empacotados e interagem entre si. Por exemplo, objetos são empacotados para encapsularem dados e processamento que manipula esses dados e interagem por meio da chamada dos métodos.
- **Propriedades não funcionais:** A descrição do projeto de arquitetura deve tratar a maneira pela qual o projeto da arquitetura interage os requisitos de desempenho, capacidade, confiabilidade, segurança, adaptabilidade e outras características do sistema que não representam as funcionalidades diretamente acessadas pelos usuários.
- **Família de sistemas:** O projeto de arquitetura deve tirar proveito de padrões reusáveis comumente encontrados no projeto de famílias de sistemas

similares. Em essência, o projeto deve ter a habilidade de reutilizar os componentes que fazem parte da arquitetura.

3.2 Padrões de Projeto na Arquitetura de Software

Para Buschmann (1995), padrões de arquitetura expressam um esquema de organização estrutural fundamental para sistemas de software.

Padrões de arquitetura são templates prontos que solucionam problemas arquiteturais recorrentes, também expressam um esquema fundamental de organização estrutural para sistemas de software, fornecendo um conjunto de subsistemas pré-definidos, especificando suas responsabilidades e incluindo regras e diretrizes para organizar as relações entre eles.

Entre os padrões de arquitetura existentes, iremos destacar dois, MVC (Model-View-Controller) e DAO (Data Access Object).

3.3 MVC (Model-View-Controller)

O MVC permite representar a comunicação entre os componentes que produzem o conteúdo de apresentação, a lógica do negócio e o processamento de requisições.

O MVC foi originalmente criado como padrão de projeto arquitetural para o ambiente Smaltalk (linguagem de programação totalmente orientada a objeto fortemente tipada), mas pode ser utilizado em qualquer tipo de aplicação.

Teruel (2012) escreve sobre MVC:

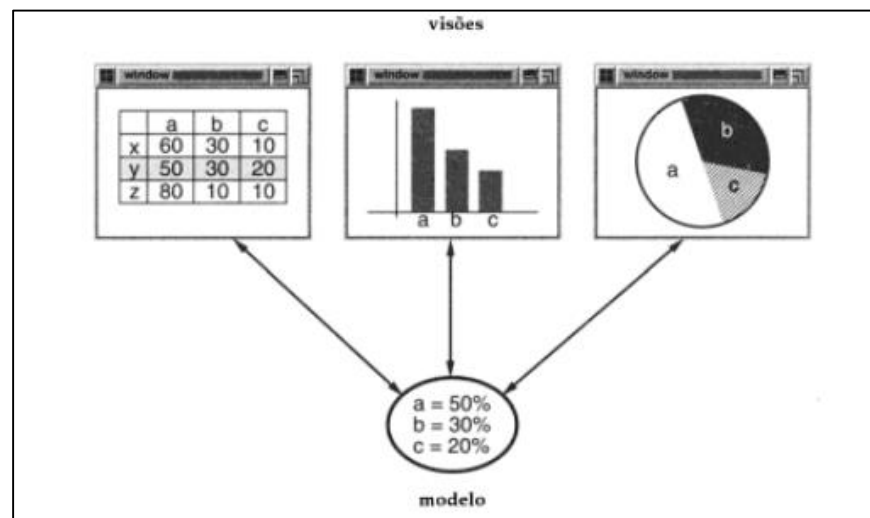
Model-View-Controller (MVC) é um padrão de arquitetura (design pattern) que pode ser utilizado para representar e entender a comunicação existente entre os componentes de uma aplicação seja para web, para desktop ou para dispositivos móveis. Os modelos de arquitetura que utilizam o padrão MVC definem claramente a separação de responsabilidade[..].

Gamma et al. (2000) descreve MVC como “A abordagem MVC é composta por três tipos de objetos. O Modelo é o objeto de aplicação, a Visão é a apresentação na tela e o Controlador é o que define a maneira como a interface do usuário reage às entradas do mesmo.”

Antes de surgir o MVC, os projetos de interfaces para o usuário agrupavam esses objetos – Model (Modelo), View (Visão) e Controller (Controle) – o conceito de MVC vem para separar esses objetos para a aumentar a flexibilidade e a reutilização de código.

Para explicar o MVC, Gamma et al. (2000) utiliza a seguinte figura (figura 3).

Figura 3 – Exemplificação do MVC



Fonte: Gamma et al. (2000, p. 21)

O seguinte diagrama mostra um modelo e três divisões (por simplificação, deixamos de fora os controladores). O modelo contém alguns valores de dados, e as visões que definem uma planilha, um histograma, e um gráfico de pizza, apresentam esses dados de várias maneiras. Um modelo se comunica com suas visões quando seus valores mudam, e as visões se comunicam com o modelo para acessar esses valores. Gamma et al. (2000)

Evandro Calos Teruel (2012) ainda defini os três grupos de componentes principais do MVC – **Model**(Modelo), **View** (Apresentação) e **Controller** (Controle):

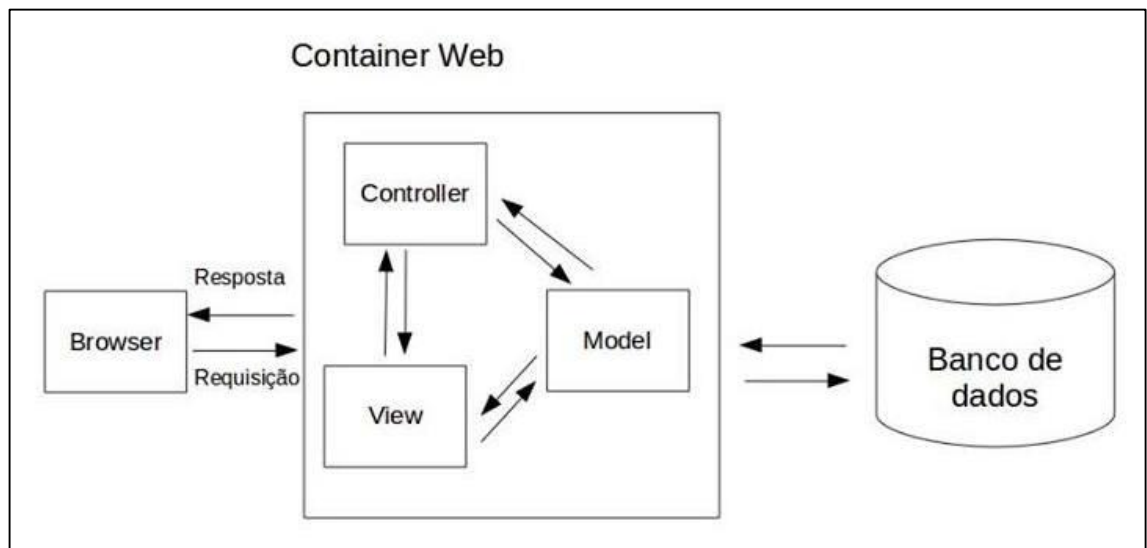
- **Model:** componente responsável pelo conhecimento do domínio de negócio. Nesse componente são representadas as classes que persistem e recuperam do banco de dados um tipo específico de objeto, classes de entidade (famosos *beans* – classes que definem um tipo específico de dados com métodos *getters* e *setters*) e as classes que implementam as regras de negócio.

- **View:** componente responsável por uma visão de apresentação do domínio de negócio. Nesse componente são apresentadas todas as classes que representam interface com o usuário, seja utilizando swing, awt, JPS etc.

- **Controller:** componente responsável por controlar o fluxo e o estado da entrada do usuário. Nesse componente são representadas as servlets (geralmente única) que fazem a comunicação entre as páginas e as classes que acessam o banco de dados. Representa a engine da aplicação.

A figura 4 representa a estrutura do padrão MVC, exemplificando o funcionamento (mais genérico) dos três grupos do MVC.

Figura 4 – Representação do fluxo de informações do padrão MVC



Fonte: Evandro Carlos Teruel (2012, p. 7)

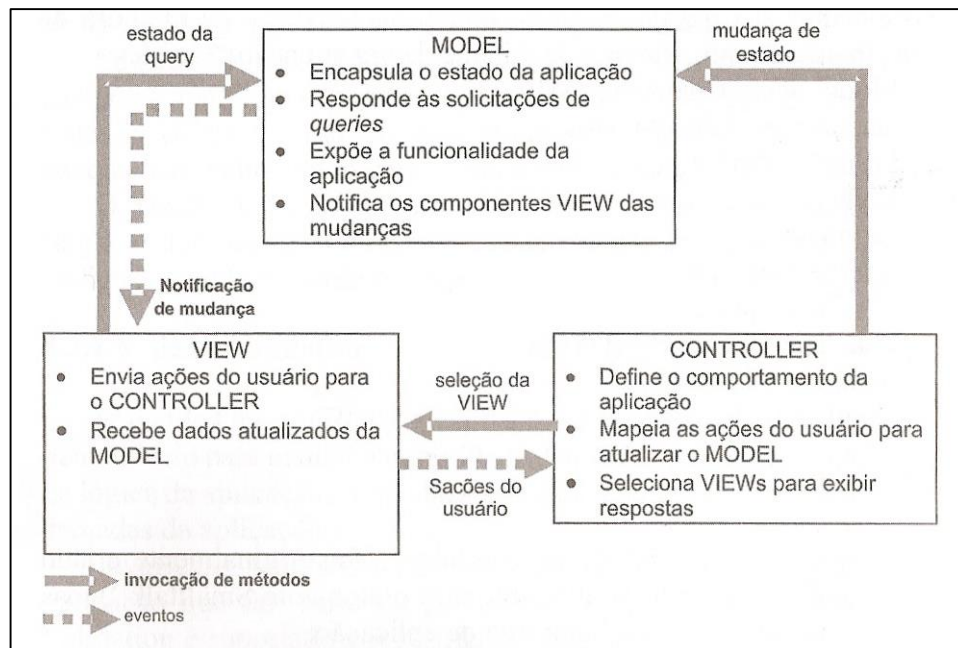
No componente View são representadas as páginas JSP, HTML e outros componentes destinados a apresentar conteúdo ao cliente.

No componente Controller são representadas as classes servlet que fazem o controle do fluxo de comunicação entre os componentes de apresentação e modelo.

No componente Model são representadas as classes *beande* entidade (aquelas que manipulam atributos por meio dos métodos *getter* e *setter*), as classes que implementam as regras de negócios e as classes que fazem a comunicação com o banco de dados e a persistência de dados nas tabelas por meio da execução de instruções SQL.

A figura 6 mostra o esquema representativo da interação entre os componentes do MVC de forma mais detalhada.

Figura 5 – Esquema representativo da interação entre os componentes do MVC



Fonte: Evandro Carlos Teruel (2012, p. 7)

3.4 DAO (Data Access Object)

Data Access Object (DAO) é um Design Pattern que permite que uma aplicação seja desenvolvida de forma que as classes da camada de acesso aos dados (componente Model do MVC) sejam isoladas das camadas superiores (componentes Controller e View do MVC).

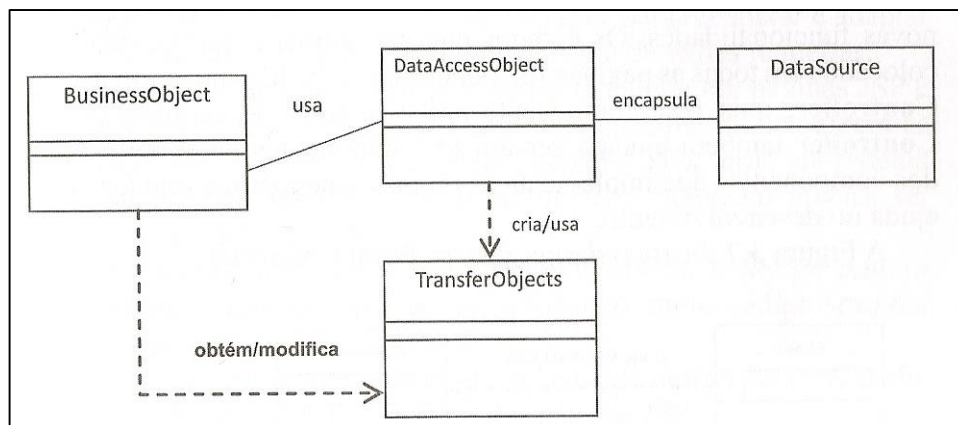
DAO representa as classes que são responsáveis por realizar a interface entre o banco de dados e o restante da aplicação. Essas classes geralmente contêm métodos básicos para persistir e recuperar dados do componente do padrão MVC: Model.

Segundo a SUN Microsystems o DAO separa um recurso de dados da interface do cliente dos mecanismos de acesso a dados e adapta uma API específica de acesso a recursos de dados para uma interface cliente genérica.

Segundo Gomes (2008), o DAO é um ponto de chamada para utilizar os recursos do banco de dados relacional, tanto para consultas (selects) quanto para inserção, alteração e exclusão (insert, update e delete). Quando se utiliza o DAO, você torna o banco de dados independente, de forma que uma mudança no banco não gera manutenção nas classes do negócio.

O padrão DAO permite que os mecanismos de acesso de dados mudem, independentemente do código que utiliza os dados.

Figura 6 – Diagrama de classes representando o design pattern DAO



Fonte: Evandro Carlos Teruel (2012, p. 117)

Transfer Objects representam os *beans*, **DataSource** representa o banco de dados, **DataAccess Object** representa os objetos de acesso a dados e **Business Object** os objetos de negócio que podem ser encarregados de iniciar a execução das operações (classes command).

De uma forma geral, é possível observar que o DAO é uma maneira de separar as regras do negócio (Business Object) do acesso a dados (Data Access Object).

3.5 Business Object (regras de negócio)

Business Object é um Design Pattern utilizado para separar os dados da lógica de negócios utilizando um modelo de objeto.

Este padrão é utilizado quando se deseja centralizar a lógica do negócio e separar o estado do negócio e comportamentos relacionados do resto da aplicação, melhorando a coesão e reutilização e aumentando a reutilização da lógica de negócios para evitar a duplicação de código.

3.6 Considerações Finais

Durante este capítulo foi abordado o que é arquitetura de software e a utilização dos padrões de projetos no desenvolvimento de software no âmbito da arquitetura.

Também foi apresentado os principais padrões de projetos da área de arquitetura de software, sua aplicação e seus impactos.

No próximo capítulo, iremos trazer resultado de um estudo de caso utilizando os estudos aplicados nos dois capítulos anteriores.

4 APLICAÇÃO DE DESIGN PATTERNS

4.1 Considerações Iniciais

Será apresentado uma empresa de desenvolvimento no qual foi realizado a utilização de Design Patterns no desenvolvimento de software.

A utilização dos padrões de projeto na estrutura do desenvolvimento de seus sistemas deve-se a necessidade de atualização do portfólio já existente, para uma plataforma mais ágil e com soluções mais atuais, permitindo que sejam aplicadas ferramentas e metodologias mais recentes e em alta no meio profissional de desenvolvimento de software.

O investimento nessa migração dos sistemas prevê uma diminuição de problemas reportados por clientes e aumento na eficiência em processos internos, durante o desenvolvimento dos sistemas.

No decorrer do capítulo abordaremos de forma mais completa as motivações para a utilização de Padrões de Projeto, os impactos de sua utilização no desenvolvimento e os resultados esperados com a implementação do Design Patterns.

4.2 Sobre a empresa

A empresa, uma das líderes brasileiras de TI para a área de logística e desenvolvedora da solução WMS (Warehouse Management System), sistema de gerenciamento de depósitos e armazéns, completou 24 anos com a marca de mais de 3.500 licenças dos aplicativos comercializados.

Com uma estrutura constituída por sua sede na capital de São Paulo e uma fábrica de software no interior do Estado, em Barra Bonita, a empresa conta com mais de 104 profissionais especializados no desenvolvimento de sistemas e no suporte aos usuários.

Detentora de um portfólio completo de produtos e serviços voltados para a automatização dos processos logísticos de diversos segmentos, a companhia tem

como foco o aperfeiçoamento e a inovação constantes para atender plenamente a todas as necessidades de cada modelo de negócio.

A desenvolvedora possui hoje 65 clientes em todo o País e mantém um crescimento médio de 28% ao ano.

4.3 Portfólio de soluções

As soluções desenvolvidas pela empresa são:

Solução WMS (Warehouse Management System): conjunto de operações automatizadas, utilizando códigos de barras, coletores de dados Rádio Frequência e Troca Eletrônica de Informações (E.D.I.), que visa otimizar o gerenciamento das atividades de Almoarifados, Depósitos e Centros de Distribuição, de forma ágil, segura e eficiente;

Solução TMS (Transportation Management System): TMS contempla um conjunto de operações que visa auxiliar o planejamento, monitoramento, controle e a execução das atividades relativas à transportes;

Solução REDEX (Recinto Especial para Despacho Aduaneiro de Exportação): A solução REDEX é um conjunto de operações informatizadas que visa otimizar o gerenciamento das atividades de exportação e armazenagem de forma ágil, segura e eficiente.

Nos anexos está disponível o detalhamento das soluções desenvolvidas pela empresa.

4.4 A busca por sustentabilidade

Na realidade atual do mercado de desenvolvimento de software, cada vez mais é uma necessidade realizar inovações em novas funcionalidade nos softwares desenvolvidos, adquirindo expertise em nichos de mercado e criando maior valor a seus clientes. Porém, além de criar softwares que atendam às necessidades de seus clientes, é necessário que empresas de desenvolvimento de software se atentem em buscar melhorias em processos internos da fábrica.

A busca por inovação nos processos de desenvolvimento de software inclui criar um portfólio de soluções que se sustentem ao longo de sua vida útil, ou seja, criar sistemas que favoreçam a manutenibilidade de seu código fonte.

No atual processo de desenvolvimento dos sistemas, uma das grandes dificuldades é a questão de realizar ajustes e customizações no código fonte, tornando o trabalho de desenvolvimento, homologação e suporte mais penosos.

IEEE (1990, p. 46, tradução livre) define manutenibilidade de software como “[...] a facilidade com que um sistema ou componente de software pode ser modificado para corrigir falhas, melhorar o desempenho ou outros atributos, ou se adaptar a uma modificação do ambiente”, seguindo essa mesma orientação, Pressman (1995) descreve a manutenibilidade como a facilidade com que um software pode ser corrigido, adaptado ou ampliado, caso haja a necessidade de inclusão ou alteração nos requisitos funcionais. Gamma et al. (2000) sustenta que os padrões de projeto têm grande influência na manutenibilidade de software. Na medida em que influenciam diretamente em alguns fatores, tais como limitação de dependências de plataforma, viabilização de estruturas em camadas, facilidade de extensão, entre outros, os padrões de projetos elevam o nível de facilidade com que um sistema de software pode ser modificado.


Os padrões de projeto também tornam uma aplicação mais fácil de ser mantida quando são usados para limitar dependências de plataforma e fornecer uma estrutura de camadas a um sistema. Eles melhoram a facilidade de extensão ao mostrar como estender hierarquias de classes e explorar a composição de objetos. (GAMMA et al., 2000, p. 41).

Para adaptar-se às mudanças necessárias para criar softwares mais eficientes, desde novembro de 2014 a empresa reservou uma parte de seus recursos para trabalharem na migração dos sistemas desenvolvidos em Delphi XE 2 e Oracle Database para a linguagem Java EE e persistência de dados multibanco.

A escolha pela linguagem Java vem de um estudo realizado no início do projeto, visando a tendência do mercado e os benefícios a longo e curto prazo. A adesão de Java como linguagem é justificada pelos gráficos abaixo:

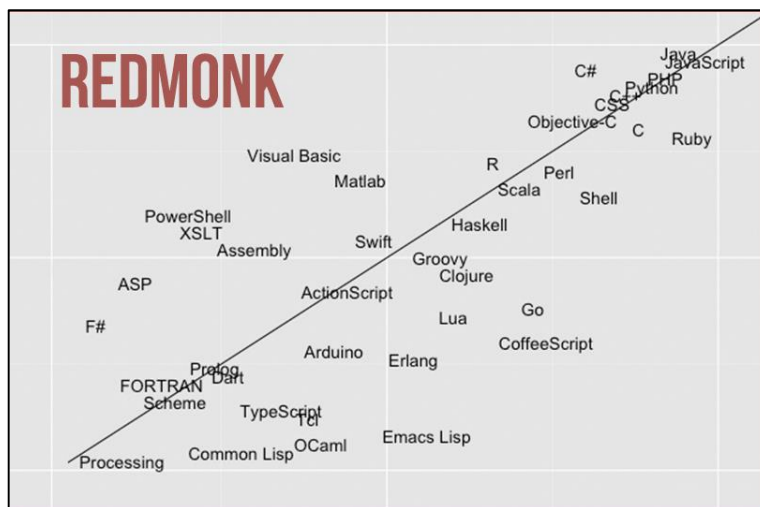
Figura 7 – Principais linguagens utilizadas no mundo – Infobase Interativa

VEJA O TOP-20 COMPLETO NO INFOGRÁFICO PRODUZIDO PELA **INFOBASE INTERATIVA** :

<p>› JavaScript</p> <p>01</p>	<p>› Java</p> <p>02</p>	<p>› PHP</p> <p>03</p>
<p>› Python</p> <p>04</p>	<p>› C#</p> <p>05</p>	<p>› C++</p> <p>05</p>
<p>› Ruby</p> <p>05</p>	<p>› CSS</p> <p>08</p>	<p>› C</p> <p>09</p>
<p>› Objective-C</p> <p>10</p>	<p>› Perl</p> <p>11</p>	<p>› Shell</p> <p>11</p>
<p>› R</p> <p>13</p>	<p>› Scala</p> <p>14</p>	<p>› Haskell</p> <p>15</p>
<p>› Matlab</p> <p>16</p>	<p>› Go</p> <p>17</p>	<p>› Visual Basic</p> <p>17</p>
<p>› Clojure</p> <p>19</p>	<p>› Groovy</p> <p>19</p>	

Fonte: Portal Tecmundo

Figura 8 – Principais linguagens utilizadas no mundo – Redmonk



Fonte: Portal Tecmundo

Como é possível observar através das pesquisas realizadas pela Redmonk e Infobase Interativa, a principal linguagem de programação utilizada no mundo é Java, seguindo a tendência mundial, a linguagem foi escolhida para se realizar a reescrita dos softwares que compõem do portfólio de soluções da empresa.

Visando a melhoria nos processos internos e principalmente melhor satisfação dos clientes, pois com a mudança proporcionadas pelo novo projeto, o cliente terá melhor performance no sistema, através de rotinas de código mais ágeis e eficientes, menor ocorrência de inconsistências (principalmente recorrência de erros) e maior agilidade na entrega de novas funcionalidades, melhorias e correções, entregando versões ao cliente em menor tempo.

4.5 Evidências sobre o projeto

Como demonstração para este projeto foi desenvolvido uma pequena aplicação utilizando os padrões de projeto Singleton, DAO, MVC, Façade e Factory Method. E para efeito de comparação, há o desenvolvimento de uma segunda aplicação desenvolvida sem utilização de padrões.

A aplicação tem como base uma agenda de contatos para Pessoas Físicas, no qual o usuário irá digitar as informações do contato e persisti-las em um banco de dados relacional.

A linguagem utilizada no projeto com padrões foi a Java SE versão 8 com a interface gráfica Swing, a IDE de desenvolvimento escolhida foi a Eclipse versão Mars. O banco de dados relacional utilizado foi o MySQL versão 5.6.

E para o projeto sem padrões foi utilizado a linguagem Free Pascal com a IDE Lazarus versão 1.4.4. Com a mesma base de dados.

Neste capítulo abordaremos a aplicação dos padrões de projetos no desenvolvimento do novo sistema projetado pela empresa com a utilização dos padrões de projeto no código fonte

4.5.1 Singleton - Garantia de unicidade simplifica o controle

Na lógica do projeto somente um usuário pode permanecer logado na aplicação e todos os cadastros feitos deverão também inserir qual foi o usuário que efetuou o cadastro.

Para a solução deste problema é utilizado o padrão Singleton, no qual ele garante uma única instância da classe no escopo da aplicação. Uma vez instanciada no log-in ela sempre retornará a mesma instancia no resto do projeto. Garantindo a utilização correta dos dados da classe.

Seu uso facilita o desenvolvimento para futuras alterações pois o desenvolvedor já sabe que a classe está devidamente instanciada e somente é necessário extrair as informações da classe.

Mas a classe que utiliza o padrão Singleton deve ser muito bem estudada e utilizada, pois ela não poderá ser utilizada dentro de loops e deve ser bem encapsulada para suas informações serem mantidas através dos processos da aplicação. E o acesso da classe é global, pois seu método de instancia é estático.

Código fonte da classe modelo Usuario.java

```
/*
 * Classe modelo para Usuário
 *
 * */
package br.com.tcc.modelo;

public class Usuario {

    // Atributos
    private int codigoUsuario;
    private String login;
    private String senha;
    private String email;
    /*
     * Singleton
     * O construtor é protegido para não ser criado em outro lugar
     * Para ser criado é necessário chamar o método getInstance
     * Se o objeto não estiver criado ele será instanciado
     * Se ela já estiver instanciada sempre retornará a mesma instância
     */
    private static Usuario instance = null;
    protected Usuario() { }
    public static Usuario getInstance() {
        if (instance == null) {
            instance = new Usuario();
        }
        return instance;
    }

    // Getters e Setters omitidos
}
```

Código Fonte da UnituntTUsuario.pas

```
// Classe do Usuário
typeTUsuario = class
public
    // Atributos
    FCodigoUsuario: Integer;
    FLogin: String;
    FSenha: String;
    FEmail: String;
    propertyCodigoUsuario :Integer read FCodigoUsuario write FCodigoUsuario;
    property Login :String read FLogin write FLogin;
    propertySenha :String read FSenha write FSenha;
    property Email :String read FEmail write FEmail;
    procedureCarregarUsuario(Conexao: TDatabase);
end;
implementation
{ TUsuario }
procedure TUsuario.CarregarUsuario(Conexao: TDatabase; sLogin, sSenha: String);
varqry: TSQLQuery;
begin
    try
        // Se forem passados parametros carrega um novo usuario
        if not (sLogin = "") and (sSenha = "") then
            begin
                Login :=sLogin;
                Senha :=sSenha;
                End;
                qry := TSQLQuery.Create(Conexao);
                qry.DataBase := Conexao;
                qry.SQL.Text := 'SELECT codigoUsuario, email FROM USUARIO WHERE LOGIN =
:LOGIN AND SENHA = :SENHA';
                qry.Params.ParamByName('Login').AsString:= Login;
                qry.Params.ParamByName('Senha').AsString:= Senha;

                qry.Open;

                ifqry.RecordCount< 1 then
                    begin
                        ShowMessage('Usuarioinválido! Digite seu login e senha novamente');
                        edtLogin.Text := "";
                        edtSenha.Text := "";
                        Exit
                    end;
```

No código em linguagem Pascal é fácil observar quais possíveis erros o desenvolvedor pode encontrar. Todos os atributos da classe estão como públicos, nos quais podem ser facilmente alterados no meio do código e as informações sobre o login podem não ser tão confiáveis se forem muito manipuladas.

Para as informações do Usuário serem mantidas seu Form não pode ser destruído enquanto a aplicação estiver sendo executada e a forma de manter e

carregar o usuário através do código é, não passar nenhum novo parâmetro no método de Carregar Usuário, caso algum desenvolvedor não conheça essa solução para o problema, caso a classe fosse mais complexa ele pode perder um tempo considerável tentando descobrir como manter a informação através da aplicação.

4.5.2 DAO - Estrutura central para persistência multibanco

Para os cadastros do sistema foi separado a manipulação dos dados em Pessoa e Pessoa Física.

Em aplicações que utilizam banco de dados para salvar informações é utilizado uma grande quantidade de scripts SQL, para recuperação, alteração e inserção de dados. Estes scripts muitas vezes ficam perdidos pelo código sendo utilizados somente quando são chamados pela aplicação, caso ele apresente mal funcionamento o desenvolvedor deverá procurar o local onde este script está sendo utilizado no código. Deixando o script SQL diretamente na classe onde é utilizado seu acesso fica muito difícil e seu reaproveitamento é quase impossível por desenvolvedores mais inexperientes no código gerando duplicidade.

O padrão DAO centraliza os repositórios de script SQL facilitando seu uso e manutenção.

Como exemplificado no código da classe PessoaDAO e PessoaFisicaDAO não foi necessário reescrever o “*insert*” na tabela “*pessoa*” na classe PessoaFisicaDAO pois ele já existe e sua utilização em qualquer outro lugar do código é muito simples.

A mesma lógica pode ser utilizada para um “*delete*” no qual o usuário pode escolher se irá deletar ambos os cadastros ou somente o cadastro da Pessoa Física.

Código fonte da Classe PessoaDAO.java

```

/*
 * Manipulação de dados da Pessoa
 *
 * */
package br.com.tcc.dao;

// Imports omitidos

public class PessoaDAO {

    Connection connection;

    public PessoaDAO() {
        connection = ConnectionMySQL.getConnection();
    }
    // Insere um registro
    public void INSERT(Pessoa p) {
        try {

            String SQL = "INSERT "
            + " INTO pessoa (codigoUsuario, nome, telefone, email, rua,"
                        + " numeroResidencia, bairro, cidade)"
            + " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

            PreparedStatement ps = connection.prepareStatement(SQL.toUpperCase());
            ps.setInt (1, Usuario.getInstance().getCodigoUsuario());
            ps.setString(2, p.getNome());
            ps.setString(3, p.getTelefone());
            ps.setString(4, p.getEmail());
            ps.setString(5, p.getRua());
            ps.setInt (6, p.getNumeroResidencia());
            ps.setString(7, p.getBairro());
            ps.setString(8, p.getCidade());

            ps.executeUpdate();
            ps.close();

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
            Logger.getLogger(UsuarioDAO.class.getName(), null);
        }
    }

    // Continuação do código

```


Código fonte da Classe PessoaFisicaDAO.java

```

/*
 * Manipulação de dados da Pessoa Fisica
 *
 */

package br.com.tcc.dao;

// Imports omitidos

public class PessoaFisicaDAO {

    Connection connection;

    public PessoaFisicaDAO() {
        connection = ConnectionMySQL.getConnection();
    }
    // Insere um registro
    public void INSERT(PessoaFisica pf) {

        try {
            // Carrega o Código Pessoa para inserir a Pessoa Fisica
            if (pf.getCodigoPessoa() == 0) {

                PessoaDAO pd = DAOFactory.createPessoaDAO();
                pd.INSERT(pf);
                // Grava o código criado para o cadastro da Pessoa Física
                int cod = pd.SELECT(pf).get(0).getCodigoPessoa();
                pf.setCodigoPessoa(cod);

            }

            String SQL = "INSERT "
                + " INTO pessoaFisica (codigoPessoa, codigoUsuario, rg,"
                + "                                cpf, cnh, tituloEleitor) "
                + " VALUES (?, ?, ?, ?, ?, ?)";

            PreparedStatement ps = connection.prepareStatement (SQL.toUpperCase());

            ps.setInt(1, pf.getCodigoPessoa());
            ps.setInt (2, Usuario.getInstance().getCodigoUsuario());
            ps.setString(3, pf.getRg());
            ps.setString (4, pf.getCpf());
            ps.setString (5, pf.getCnh());
            ps.setInt (6, pf.getTituloEleitor());

            ps.executeUpdate();
            ps.close();

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
            Logger.getLogger(UsuarioDAO.class.getName(), null);
        }

    }

    // Continuação do código

```

Código Fonte da Unit untFrmCadPessoaFisica.pas

```
// Se for para inserir
ifacao = INSERIR then
begin
    // Insere a pessoa
    sql.Append('INSERT INTO pessoa');
    sql.Append('(codigoUsuario, nome, telefone, email, ');
    sql.Append('rua, numeroResidencia, bairro, cidade)');
    sql.Append('VALUES (:CODIGOUSUARIO,');
    sql.Append(':NOME');
    sql.Append(', :TELEFONE');
    sql.Append(', :EMAIL');
    sql.Append(', :RUA');
    sql.Append(', :NUMERO');
    sql.Append(', :BAIRRO');
    sql.Append(', :CIDADE');
    // Finaliza o insert
    sql.Append(')');

    // Prepara os parametros
    // Executa o SQL

    // select para pegar o codigo da pessoa
    sql.Append('SELECT codigoPessoa');
    sql.Append(' FROM pessoa p');
    sql.Append(' WHERE p.nome IS NOT NULL');
    // Continuação do SELECT omitida

    // Prepara os parâmetros
    edtcodigoPessoa.Text := qry.FieldByName('codigoPessoa').AsString;
    qry.SQL.Clear;
    sql.Clear;
    // Insere a pessoa fisica
    sql.Append('INSERT INTO pessoaFisica');
    sql.Append(' (codigoUsuario, codigoPessoa, rg, cpf, tituloEleitor, cnh)');
    sql.Append(' VALUES ( ');
    sql.Append(' :CODIGOUSUARIO');
    sql.Append(', :CODIGOPESSOA');
    sql.Append(', :RG');
    sql.Append(', :CPF');
    sql.Append(', :TITULOELEITOR');
    sql.Append(', :CNH');
    // finaliza o insert
    sql.Append(' )');
    // Prepara os parametros
    // Executa o SQL
end
```

Para inserir um novo registro no projeto em Pascal toda a lógica e validações estão juntas em um único procedimento, cenário bastante comum para desenvolvedores veteranos. Muitos sistemas legado funcionam assim, uma das causas que sua manutenção é trabalhosa e cara.

Caso o desenvolvedor necessite adicionar algum campo novo no banco de dados é fácil notar a dificuldade em alterar todo o código espalhado e duplicado através da unit. E sua reutilização fica muito trabalhosa ou quase impossível, caso o código tivesse validações no meio do scrit de SQL tentar copiar o código e colar em outra unit pode se mostrar muito trabalhoso pois o desenvolvedor terá que analisar o que deverá ser deixado de lado ou o que tem que ficar.

Comparando com o código da classe PessoaFisicaDAO.java é fácil notar quanto retrabalho foi extinguido graças ao DAO da PessoaDAO.java, sendo necessário somente chamar o método inserir da PessoaDAO.java.

Ou se tivermos que incluir um novo recurso ao sistema como incluir uma Pessoa Jurídica com o DAO já temos todo o scrit SQL feito e testado para a abstração Pessoa. Sem o padrão teremos muito trabalho para organizar ou refazer o código e tudo terá que ser testado novamente, pois ele foi retirado de um ambiente e posto em outro.

4.5.3 Façade – Simplificação de uma biblioteca de código

Quando o código possui muitas classes e objetos para serem utilizados no código sua referência fica trabalhosa, dificulta a leitura e entendimento do código, o que gera tempo perdido desvendando o funcionamento e procurando o caminho dos métodos utilizados.

Além de referenciar objetos em uma classe, futuramente esses mesmos objetos deverão ser referenciados em outra, gerando retrabalho.

Com o padrão Façade tudo isso fica centralizado em um objeto em que todas as pré-configurações já estão prontas.

O padrão simplifica o uso de várias bibliotecas em um único ponto, como no exemplo em que para validar e inserir um registro no banco é só necessário chamar

o método do façade, sendo desnecessário o desenvolvedor se preocupar em criar todas as referências para os objetos e chamar cada método individualmente.

Código Fonte da Classe FacadePessoa.java

```
public static boolean Inserir(PessoaFisicaf) {  
  
    PessoaFisicaDAO pfd = DAOFactory.createPessoaFisicaDAO();  
    ValidarPessoa vp = new ValidarPessoa();  
  
    // Faz as validações dos valores inseridos  
    if (vp.isPessoaValida(pf)) {  
        // Se for válido insere uma nova PessoaFisica  
        pfd.INSERT(pf);  
        // Pode reconfigurar a tela  
        return true;  
    } else {  
        // Permanece sem alterações na tela  
        return false;  
    }  
}  
// Continuação do código
```

Código Fonte da Unit untFrmCadPessoaFisica.pas

```

procedure TfrmCadPessoaFisica.btnSalvarClick(Sender: TObject);
var sql: TStringList;
    Usuario: TUsuario;
begin
    sql := TStringList.Create;
    Usuario := TUsuario.Create;
    qry.Close;
    try
        // Carrega o Usuario
        Usuario.CarregarUsuario;

        // Verifica se foi inserido o nome
        if edtNomePessoa.Text = '' then
            begin
                ShowMessage('É necessário inserir um nome para o Contato!');
                edtNomePessoa.SetFocus;
                Exit;
            end;

        // Verifica o Telefone
        if (StrToIntDef(edtTelefone.Text, -1) = -1) and
            not (edtTelefone.Text = '') then
            begin
                ShowMessage('Insira um número válido para o Número de Telefone');
                edtTelefone.Text := '';
                edtTelefone.SetFocus;
                Exit;
            end;

        // Verifica o número de residência
        if (StrToIntDef(edtNumero.Text, -1) = -1) and
            not (edtNumero.Text = '') then
            begin
                ShowMessage('Insira um número válido para o Número de Residência');
                edtNumero.Text := '';
                edtNumero.SetFocus;
                Exit;
            end;

        // Continuação da lógica do botão Salvar
    
```

Um cenário clássico em desenvolvimento é decidir onde uma validação será adicionada. Sem seguir nenhum padrão em Pascal elas foram adicionadas antes dos scripts SQL, mas o problema é que a lógica está toda agrupada em uma mesma unit. Todas as validações tanto da Pessoa quanto da Pessoa Física estão no mesmo lugar, inviabilizando sua reutilização.

Se for necessário utilizar a mesma validação em outro local do código teremos que copiar o código e transportá-lo para onde queremos, gerando código duplicado. Agora se tivermos que alterar alguma coisa em ambas as validações teremos trabalho em dobro.

Com o padrão Façade todas as chamadas de métodos estão divididas e podem ser facilmente acessadas pelo desenvolvedor, que não terá trabalho decidindo em qual local deverá adicionar mais validações ou chama-las em qualquer outro local do código.

4.5.4 MVC – Divisão de tarefas e responsabilidade

Não é difícil encontrar soluções no qual quase toda a lógica da operação encontra-se em seus componentes, validações dos campos, cálculos e manipulação do banco de dados.

Aninhar muita lógica em um único local dificulta seu entendimento, alteração e reutilização, até chegar no ponto na qual fazer uma simples alteração causa um grande impacto em seu funcionamento.

O padrão Model, ViewController é um padrão estrutural que auxilia o desenvolvedor a organizar o seu código de maneira simples e de fácil entendimento facilitando alterações em qualquer camada da aplicação.

Ele separa a aplicação em uma camada que fica responsável por apresentar as informações em campos de texto, grids ou gráfico, essa parte é conhecida como front-end (View). As informações apresentadas são armazenadas em classes modelo (Model) no qual sua função é somente armazenar os dados, a regra do negócio, cálculos e validações é feita pelo controlador (Controller).

É um padrão muito utilizado no mundo Web, mas pode ser aplicado para qualquer outra plataforma.

Um de seus problemas é que o time deve compreender corretamente onde inserir cada parte da lógica do programa para não desestruturar o padrão, mas quando o time possui conhecimento sobre sua estrutura o desenvolvimento ocorre com muito mais facilidade.

Código fonte da Classe FacadePessoa.java

```
/*
 * Classe modelo para Pessoa Fisica
 * Filha de Pessoa
 *
 */
package br.com.tcc.model;
public class PessoaFisica extends Pessoa {

    // Atributos
    private int codigoPessoaFisica;
    private String rg;
    private String cpf;
    private String cnh;
    private int tituloEleitor;

    // Construtor
    public PessoaFisica() {
        this.rg = "";
        this.cpf = "";
        this.cnh = "";
        this.tituloEleitor = 0;
    }

    // Getters e Setters omitidos
}
```

Código fonte da Classe View CadastroPessoaFisica.java

```
if (acao == INSERIR) {  
    try {  
        pf = FacadePessoa.CarregaPessoaFisica(0,  
                                                0,  
                                                edtNome.getText(),  
                                                edtTelefone.getText(),  
                                                edtEmail.getText(),  
                                                edtRua.getText(),  
                                                edtNumero.getText(),  
                                                edtBairro.getText(),  
                                                edtCidade.getText(),  
                                                edtRg.getText(),  
                                                edtCpf.getText(),  
                                                edtTituloEleitor.getText(),  
                                                edtCnh.getText());  
        // Faz as validações e insere o novo registro  
        if (FacadePessoa.Inserir(pf)) {  
            // Altera o estado dos botões  
  
            ControleBotões(SALVAR);  
            // Recarrega a tabela  
            CarregarTabela(dtm);  
        }  
    } catch (NumberFormatException) {  
        JOptionPane.showMessageDialog(null, "Número informado inválido");  
    } catch (RuntimeException) {  
        JOptionPane.showMessageDialog(null, e.getMessage());  
    }  
}
```


Código fonte da Classe Controller ValidarPessoa.java

```

/*
 * Classe Controller para Pessoa
 *
 */
package br.com.tcc.controller;
//Imports omitidos
public class ValidarPessoa {

    // Criação das classes de apoio
    ValidacaoBasica vb = new ValidacaoBasica();
    MensagemDeErro mde = new MensagemDeErro();

    public boolean isPessoaValida(Pessoa p) {
        try {

            // Verifica se foi informado nome
            if (p.getNome().equals("")) {
                JOptionPane.showMessageDialog(null, mde.NomeNulo());
                return false;
            }

            // Verifica se o telefone informado é válido
            if (!p.getTelefone().equals("")) {
                if (!vb.validarSomenteNumero(p.getTelefone())) {
                    JOptionPane.showMessageDialog(null, mde.TelefoneInvalido());
                    return false;
                }
            }

            // Verifica se o email informado é válido
            if (!p.getEmail().equals("")) {
                if (!p.getEmail().contains("@") && p.getEmail().contains(".com"))
                    JOptionPane.showMessageDialog(null, mde.TelefoneInvalido());
                return false;
            }

            // Verifica se foi informado uma forma de contato
            if (p.getEmail().equals("") && p.getTelefone().equals("")) {
                JOptionPane.showMessageDialog(null, mde.InformacaoContato());
                return false;
            }

            // Informações OK
            return true;

        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
            return false;
        }
    }
}

```

Todo o código em Pascal apresentado até o momento está acoplado aos cliques dos botões do formulário.

É fácil visualizar todos os problemas que essa abordagem pode causar, se dois desenvolvedores começarem a dar manutenção na mesma unit, um na parte gráfica e outro na parte lógica ambas alterações irão influenciar no trabalho do outro.

No final das alterações todas as alterações deverão ser unidas e disponibilizadas, mas e se ambos alteraram o mesmo local? Mais retrabalho corrigindo erros que poderiam não existir se cada parte do código fizesse seu papel específico, como é no padrão MVC.

Cada desenvolvedor pode alterar o código sem medo de encontrar alguma alteração que possa impactar gravemente suas alterações.

4.5.5 FactoryMethod – Criação de objetos simplificada

Quando um sistema possui muitos objetos sua criação pode se apresentar problemática pois o desenvolvedor deve conhecer quais classes referenciar corretamente.

O padrão Factory Method facilita a criação de objetos criando uma interface de fácil acesso no qual as subclasses decidem qual objeto será criado. Facilitando o uso dos objetos do sistema.

Código fonte da Classe DAOFactory.java

```
/*
 * Factory Method das classes DAO
 *
 * Para manipulação dos dados
 * As classes DAO deverão ser instanciadas pela fábrica
 */
package br.com.tcc.factory;

// Imports omitidos

public class DAOFactory {

    // Manipulação dos dados da tabela Usuario
    public static UsuarioDAO createUsuarioDAO() {
        return new UsuarioDAO();
    }

    // Manipulação dos dados da tabela Pessoa
    public static PessoaDAO createPessoaDAO() {
        return new PessoaDAO();
    }

    // Manipulação dos dados da tabela PessoaFisica
    public static PessoaFisicaDAO createPessoaFisicaDAO() {
        return new PessoaFisicaDAO();
    }

}

// Continuação do código
```

No projeto em Pascal somente foi criada uma classe para o Usuário, diferente do projeto em Java que temos várias classes.

Mas é fácil notar como o uso do padrão iria deixar muito mais produtivo o desenvolvimento e manutenção do código da unit em Pascal, criando classes para a Pessoa, Pessoa Física e separar as validações em outra classe.

E utilizar o padrão para criar uma interface de criação dessas classes, facilitando o uso de todas elas para todos os outros desenvolvedores.

4.6 Resultados esperados

Conforme tratamos no capítulo 4.2 – Motivação, o principal motivo para o início deste projeto, foi a necessidade do aumento da manutenibilidade do código

fonte dos sistemas. Com isso, iria aumentar a qualidade e eficiência dos processos internos da fábrica e aumentaria a satisfação dos usuários do software.

Através de entrevistas realizadas com a equipe de desenvolvida, que está diretamente envolvida no projeto, foram definidos os principais resultados que se esperam atingir na conclusão do projeto.

Entre os resultados esperados, voltados à área técnica da programação, é ter rotinas de códigos mais eficientes, ágil, tornando processamentos mais velozes. Conforme foi visto nos capítulos no qual estudamos a finalidade da utilização dos padrões de projetos na engenharia de software e arquitetura de software, ela está estreitamente ligada ao reuso do código fonte, contribuindo para uma rotina de software mais enxuta e de alta coesão.

Com esse projeto, há como objetivo principal a melhoria da visão que a empresa passa para os usuários dos softwares desenvolvidos pela empresa. De forma mais concreta, espera-se que o usuário seja atingido pela melhoria em maior performance nos processos realizados pelos sistemas, assim aumentando a performance e eficiência dentro das empresas dos clientes (usuários) dos sistemas.

Outro importante objetivo que se tem como prioridade, é a diminuição de recorrência de inconsistências de rotinas nos softwares, pois a geração de versões de software que possuem problemas causa uma grande perda de tempo e energia das equipes e diminui a satisfação dos clientes.

Através da diminuição de inconsistências em versões de softwares finalizadas, consequentemente aumentará a qualidade e a velocidade de softwares gerados, assim conseguirá entregar versões com ajustes e melhorias aos clientes com mais velocidade e maior índice de qualidade.

4.7 Desenvolvimento do Projeto

Para a obtenção dos dados apresentados no estudo da aplicação do Design Patterns no desenvolvimento de software, foram realizadas reuniões com uma equipe de cinco pessoas reservadas pela empresa para o desenvolvimento deste novo projeto.

Durante o desenvolvimento deste estudo foram apresentadas perguntas à esta equipe, através de questionário com perguntas diretas sobre o projeto, tratando de perguntas técnicas sobre a aplicação dos padrões de projetos e sobre os resultados esperados para a empresa e os benefícios para todos os demais envolvidos no projeto – diretamente e indiretamente.

O código fonte do novo projeto desenvolvido pela empresa, por se tratar de propriedade intelectual da empresa, não foi disponibilizado neste trabalho. Então, para representar a aplicação dos padrões de projetos no código fonte, nós realizamos o desenvolvimento de um software – descrito nas evidências sobre o projeto – com a aplicação dos padrões de projetos, representando a aplicação destes padrões pela empresa de desenvolvimento de software.

5 CONCLUSÃO

Padrões de projeto para o desenvolvimento de software surgiram para a solução de problemas recorrentes no desenvolvimento de sistemas. Com base nos estudos bibliográficos realizados para o desenvolvimento deste trabalho, foi possível observar que a adoção do Design Patterns no desenvolvimento de software traz benefícios comprovados por estudiosos da área e os resultados da utilização de padrões de projetos são impactantes nos resultados finais em projetos de sistemas de softwares.

A utilização de Padrões de Projetos no desenvolvimento de software vem sendo uma ferramenta altamente necessária para as indústrias da área de desenvolvimento se manterem competitivas no mercado, pois a utilização do Design Patterns proporciona maior agilidade nos processos de desenvolvimento do portfólio das fábricas de software.

Os benefícios da implementação dos padrões de projetos são os principais motivadores para a adoção dos mesmos, que são aumento da manutenibilidade do código fonte, facilitada no entendimento do código desenvolvido, menor tempo de desenvolvimento, ganho de performance das rotinas desenvolvidas, diminuição de inconsistências e de reincidências. Todos esses benefícios trazidos pela adoção do Design Patterns, aumentam a competitividade de uma companhia desenvolvedora de software, assim tornando-a sustentável e agradando seus clientes.

Além dos estudos teóricos, conseguimos ver o impacto e a necessidade, no cenário atual do desenvolvimento de softwares, a utilização de Design Patterns através do estudo de caso realizada com uma empresa de desenvolvimento. Através deste estudo de caso, foi possível constatar que os motivos da utilização do Design Patterns em projetos de software levantados nos estudos teóricos realmente se aplicam no mercado.

O estudo de caso realizado nos mostra a necessidade uma constante atualização dos métodos de trabalhos na área de programação, pois conforme foi apresentado durante a motivação do projeto de implementação dos padrões de projetos, a empresa realizou um grande estudo de mercado, para avaliar a viabilidade da implementação desta ferramenta, dispendendo de recursos e tempo

para investir em preparação para migrar todo o seu portfólio para uma nova linguagem de programação compatível com a necessidade do Design Patterns.

Através das evidências desenvolvidas para efeito de comparação do “antes” e “depois” da utilização dos padrões de projeto, é possível constatar o impacto de seu uso no código, no qual fica visível a melhoria proporcionada pelo seu uso no desenvolvimento, tornando o código fonte do sistema mais eficiente e consistente.

Com o desenvolvimento desta monografia, foi apontado a necessidade de inovação constante nas indústrias de desenvolvimento de software, no qual o Design Patterns, objeto de estudo deste trabalho, demonstra a necessidade dessa inovação.

Durante os estudos realizados para este trabalho, conseguimos aplicar conhecimentos obtidos na teoria e visualizar os resultados da aplicação destes conhecimentos em um ambiente profissional, e com essa aplicação conseguimos mensurar o impacto que uma implementação de Design Patterns causa, tanto com pontos positivos já citado e pontos negativos.

Pontos negativos são levantados como necessidade de um planejamento muito bem detalhado de todos os passos que serão necessários serem realizados para iniciar um projeto utilizando Padrões de Projeto. Esse planejamento é desgastante por necessitar de recursos, como pessoas, tempo e investimento financeiro para capacitação de desenvolvedores e contratação de pessoas especialistas no assunto para passar treinamentos.

Porém, durante o acompanhamento dos resultados no período em que foi acompanhada a operação de mudança do portfólio, foi visto que todo o investimento realizado pela empresa obteve o retorno esperado.

Tivemos como objetivo principal deste trabalho de conclusão de curso conseguir documentar o papel dos Padrões de Projetos no desenvolvimento de softwares, realizando o estudo da aplicação dos conhecimentos através da literatura sobre o assunto e comparação com aplicação do Design Patterns em um ambiente de produção de softwares. Por fim concluímos que o objetivo principal deste trabalho foi atingido, no qual conseguimos ter, com sucesso, os resultados que esperamos.

BIBLIOGRAFIA

ALEXANDER, C. **The timeless way of building**. New York: Oxford University Press, 1979.

APPLETON, B. (1997). Patterns and software: Essential concepts and terminology.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. São Paulo: Bookman, 2000.

HAMANN, Renan. **Quais as linguagens de programação mais usadas atualmente?** 2015. Disponível em <<http://www.tecmundo.com.br/programacao/82480-linguagens-programacao-usadas-atualmente-infografico.htm>>

PRESSMAN, ROGER S. **Engenharia de Software: Uma abordagem Profissional**. Porto Alegre: AMGH Editora Ltda, 2011.

TERUEL, EVANDRO CARLOS. **Arquitetura de Sistemas: Para WEB com Java utilizando Design Patterns e Frameworks**. Rio de Janeiro, 2012.

ANEXO A

Portfólio empresa Estudo de Caso:

Solução WMS (Warehouse Management System): A solução WMS é um conjunto de operações automatizadas, utilizando códigos de barras, coletores de dados Rádio Frequência e Troca Eletrônica de Informações (E.D.I.), que visa otimizar o gerenciamento das atividades de Almoxarifados, Depósitos e Centros de Distribuição, de forma ágil, segura e eficiente.

Esta solução, totalmente parametrizada e integrada ao sistema corporativo, recebe informações (dos módulos de Recebimento, Faturamento e Estoques), processa-as em seu devido tempo e gera informações para atualizar as bases de dados da sua empresa.

Figura 9 – Características do WMS

PARAMETRIZAÇÃO, MODULARIDADE E FLEXIBILIDADE	CUSTOMIZADO DE ACORDO COM AS NECESSIDADES DO CLIENTE	FOCADO PARA TODAS AS MODALIDADES LOGÍSTICAS DE ARMAZENAGEM
SUGESTÃO DE ARMAZENAGEM E EXPEDIÇÃO	INTEGRAÇÃO COM O SISTEMA ERP POR MEIO DE EDI	SEGURANÇA E CONTROLE DE ACESSOS
MULTI- ESTABELECIMENTO	MULTIDEPÓSITO	RASTREABILIDADE DE MERCADORIAS
CONTROLE DE DADOS LOGÍSTICOS	CONTROLE DE QUALIDADE DOS ESTOQUES	INVENTÁRIO
ALTA PERFORMANCE DE ACESSO E PROCESSAMENTO	INTERFACE GRÁFICA AMIGÁVEL	FACILIDADE NA GERAÇÃO DE RELATÓRIOS (CRYSTAL REPORTS)
USO DE COLETORES DE DADOS POR RA- DIOFREQUÊNCIA (RF)	HISTÓRICO DE ATIVIDADES POR USUÁRIO/ ATIVIDADE	GERENCIAMENTO DE KITS

Fonte: www.storeautomacao.com.br

Os módulos do WMS são divididos em:

1. Automação

COLETOR visa o gerenciamento de conexões de coletores de dados de RF (Rádiofrequência). Pode gerenciar simultaneamente diversas conexões de diferentes marcas e tipo de coletores de dados, além de permitir que o usuário possa reportar em tempo real todos os processos internos, incrementando a acuracidade dos estoques. Pode atuar também no modo pró-ativo (em conjunto com convocação).

CONVOCAÇÃO (PRÓ-ATIVO) detecta uma necessidade, avalia os parâmetros existentes e convoca o operador que esteja disponível e autorizado para executar aquela operação.

MONITOR E SCHEDULE permite agendar determinados processos para rodarem automaticamente de tempos em tempos, tais como: as parametrizações dos processos que serão executados, os horários e as frequências que ocorrerão.

2. Visualização do depósito

DESIGNER ferramenta gráfica utilizada para criar, manter e visualizar a planta de um estabelecimento que faça armazenagem de produtos (armazém, almoxarifado, depósito ou centro de distribuição). Além de ser possível visualizar os percentuais de ocupação.

3. Integrações

COMNECT sistema para troca eletrônica de informações, permite geração/leitura de arquivos para integração entre sistemas próprios ou de terceiros (EDI).

ORAINTE realiza a integração entre sistemas com banco de dados Oracle, utilizando, para isso, um banco intermediário.

4. Business Intelligence

B.I. geração de gráficos e painel de controle para acompanhamento das operações.

5. Consultas Web

WMS_NET módulo de visualização de relatórios do sistema pela web, em tempo real. Permite ao gestor do armazém/cliente depositante controlar as informações fornecidas pelos relatórios de qualquer lugar que possam estar – desde que possua alguma conexão com a internet

6. Gestão de entregas

- Controle de mercadorias em trânsito;
- Identificação do local de entrega;
- Rastreamento do veículo (GPS);
- Rotas;
- Monitoramento dos tempos;
- Produtividade;
- Gestão de avarias na entrega;
- Conferência de pedidos.

7. Faturamento de serviços

FATURAMENTO disponibiliza diversas formas de realizar o faturamento, sendo possível escolher entre 40 regras cadastradas no sistema ou customizar uma regra de acordo com a particularidade da operação.

8. Emissão de livros fiscais

FISCAL controle fiscal das operações realizadas, contando inclusive com Livro Fiscal de Armazém Geral.

9. Nota fiscal eletrônica

NF-e emissão de Nota Fiscal Eletrônica, por meio da parceria com a conceituada NDdigital.

Solução TMS (Transportation Management System): TMS contempla um conjunto de operações que visa auxiliar o planejamento, monitoramento, controle e a execução das atividades relativas à transportes.

Possui 3 módulos distintos e independentes:

TMS/Transportador: voltado para transportadores que realizam o serviço de frete.

TMS/Embarcador: voltado para embarcadores que contratam o serviço de frete.

TMS/Pneus: voltado para o controle, gerenciamento e rastreabilidade de pneus dos veículos.

Figura 10 – Características do TMS



Fonte: www.storeautomacao.com.br

Solução REDEX (Recinto Especial para Despacho Aduaneiro de Exportação): A solução REDEX é um conjunto de operações informatizadas que visa otimizar o gerenciamento das atividades de exportação e armazenagem de forma ágil, segura e eficiente.

Recebe, processa e integra informações de diversas entidades às bases de dados do sistema corporativo. Além disso, cobre operações de exportação (envolvendo mercadorias e contêineres) e atende a todos os tipos de modalidades de transporte (marítimo, rodoviário, aéreo e ferroviário).

Figura 11 – Características do REDEX



Fonte: www.storeautomacao.com.br

A estrutura de departamentos da empresa conta com equipes de desenvolvimento (WMS, Coletor, REDEX e Integrações), suporte para atendimento ao cliente e setor de qualidade para homologação dos sistemas.

ANEXO B

Questionário “Implementação Padrões de Projeto” aplicado à equipe do projeto da empresa objeto do estudo de caso:

1) Qual sua designação no projeto.

Resposta: Analista Programador

2) Qual a motivação principal para se aplicar padrões de projeto no novo software.

Resposta: Manutenibilidade

3) Quais as principais mudanças na produção de código depois de implementar padrões.

Resposta: Rotinas enxutas, reuso de código, alta coesão e baixo acoplamento.

4) Pergunta: Quais os padrões mais utilizados e quais padrões não deram certo.

Resposta: Inversion of Control (IoC), Dependency Injection (DI), Factory, Builder, Strategy, Observer, Template Method, MVC (Model-View-Controller).

Não houve nenhum padrão que não deram certo.

5) Dificuldades encontradas ao implementar os padrões.

Resposta: Entendimento correto do padrão a ser implementado.

6) Resultados esperados.

Resposta: Padronização, reuso de código e facilidade de manutenção.

7) Número de funcionários alocados para o projeto, data inicial e data prevista para conclusão.

Resposta:

Etapa de Pesquisa e Desenvolvimento

- Funcionários: 5.
- Data inicial: Nov. / 2014.
- Data conclusão: Out/2015.

Etapa de Desenvolvimento do Produto:

- Funcionários: 19.
- Data inicial: Jan/2016.
- Data prevista para conclusão: Ainda não definida.
- Mudanças previstas para o usuário do software.

Resposta:

- Maior performance.
- Menor recorrência de bugs.
- Maior agilidade na entrega de novas funcionalidades e melhorias.
- Dúvidas no início do projeto.

Resposta: Dúvidas sobre arquitetura, tecnologia, organização do projeto e padrões a serem utilizados.

- Quais foram os requisitos necessários para dar início ao projeto.

Resposta:

- Domínio dos padrões de projeto.
 - Definição da arquitetura do projeto.
 - Definição das tecnologias utilizadas.
-
- Quais foram os desafios previstos para os futuros desenvolvedores que iniciarem no desenvolvimento seguindo os padrões.

Resposta: Entendimento correto dos padrões que devem ser implementados. Através de revisão de código, orientar os desenvolvedores a seguirem os padrões propostos pela arquitetura definida.