MARCOS HIDALGO NUNES

DEFINIÇÃO DE PROCESSO PARA CONSTRUÇÃO DE SISTEMAS BASEADO EM *FRAMEWORKS* DE DESENVOLVIMENTO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Leonardo Dominguez Dias

São Paulo
2012

FICHA CATALOGRÁFICA

Nunes, Marcos Hidalgo
Definição de processo para construção de sistemas baseado em frameworks de desenvolvimento / M.H. Nunes. -- São Paulo, 2012.
63 p.

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Sistemas e processos construtivos 2. Frameworks I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

FICHA CATALOGRÁFICA

Nunes, Marcos Hidalgo
Definição de processo para construção de sistemas baseado em frameworks de desenvolvimento / M.H. Nunes. -- São Paulo, 2012.
63 p.

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Sistemas e processos construtivos 2. Frameworks I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

DEDICATÓRIA

Dedico este trabalho à minha família que me apoiou desde o início da minha aprendizagem.

AGRADECIMENTOS

Ao orientador Leonardo Dominguez Dias pela orientação e o interesse pelo sucesso deste trabalho.

Aos meus pais que me deram as condições para atingir meus feitos educacionais e profissionais.

Às minhas irmãs que me apoiaram no que puderam para que eu escrevesse este trabalho.

Aos amigos e colegas de trabalho que direta ou indiretamente ajudaram na execução deste trabalho.

RESUMO

Embora haja uma extensa documentação sobre os benefícios do reuso de código, usando técnicas como *design patterns*, essa prática é menos adotada que o esperado. A realidade encontrada nas organizações é que os aspectos de desempenho ou manutenabilidade não têm a mesma importância que o cumprimento dos prazos de entrega. Essas concessões em relação à qualidade do código acabam refletindo em custos futuros para manter essas aplicações em funcionamento. Tendo em vista esse quadro, o presente trabalho propõe a criação de um processo para construção de sistemas baseado no uso *frameworks* de desenvolvimento construídos via aplicação de *design patterns*. Nele é feito um estudo de caso em uma empresa que deseja estabelecer padrões para construção de sistemas de informação. O resultado demonstra que a reutilização de código permite ao mesmo tempo melhorar a qualidade e reduzir o tempo de desenvolvimento de aplicações.

ABSTRACT

Although there is an extensive documentation about the benefits of code reuse using techniques such as design patterns, this practice is less adopted than expected. Reality in organizations shows that aspects such as performance or maintainability are less important than the compliance with deadlines. These concessions regarding the code quality are reflected on future costs to keep those applications running. Given this framework, this paper proposes the creation of a process for building systems based on the use of development frameworks constructed via the application of design patterns. It is considered a case study in a company that intends to establish standards for building information systems. Result demonstrates that the code reuse permits both the quality improvement and to reduce the time for the development of applications.

LISTA DE ILUSTRAÇÕES

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
BI	*Business Intelligence*
BPMN	*Business Process Modeling Notation*
CMMI	*Capability Maturity Model Integration*
GoF	*Gang of Four* (Erich Gamma *et. al.*)
IDE	*Integrated Development Environment*
ISO	*International Standards Organization*
MOO	Metodologia Orientada a Objetos
MPS.BR	Melhoria de Processos do Software Brasileiro
POO	Programação Orientada a Objetos
QA	*Quality Assurance*
SGBD	Sistema Gerenciador de Banco de Dados
TDD	*Test-Driven Development* (desenvolvimento orientado a testes)
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
XP	*eXtreme Programming*

SUMÁRIO

1. INTRODUÇÃO

1.1. Considerações Iniciais

A construção de sistemas de informação baseia-se no uso de linguagens de programação, que permitem ao desenvolvedor transformar os códigos-fonte escritos nessas linguagens em códigos de máquina executadas por computadores.

As linguagens de programação utilizam um conjunto finito de sentenças, mas que oferecem uma ampla possibilidade de combinações. Essas variações são decorrentes da lógica utilizada pelos desenvolvedores na codificação de software. Dessa forma há a possibilidade de chegar ao mesmo resultado de maneiras diferentes.

Com o tempo, algumas lógicas de programação comprovaram sua eficiência na resolução de problemas específicos e que se repetem em diferentes cenários. A compilação desses resultados em um corpo de conhecimento levou ao conjunto de padrões conhecido por *design patterns* (APPLETON, 1997).

Design patterns facilitam o reuso de modelos bem conhecidos e sucedidos e oferecem uma arquitetura baseada na experiência de especialistas. Eles também ajudam desenvolvedores a escolherem alternativas que tornam um sistema reusável e evitam outras que comprometam esse reuso (CHANG, LU, & HSIUNG, 2011).

1.2. Objetivo

Este trabalho propõe a criação de um processo para sistematizar o uso de boas práticas de programação por um conjunto de desenvolvedores com diferentes formações e experiências profissionais. Ele é estabelecido pela construção de um

framework de desenvolvimento baseado no uso de *design patterns*. Dessa forma, espera-se que a codificação seja feita de maneira uniforme, utilizando adequadamente os recursos computacionais de uma organização e produza softwares que atendem às necessidades de negócio.

O desenho do processo é feito por meio da técnica de instanciação, aplicando conceitos descritos por Panigassi (2007) e Dias (2010) ao cenário de definição de padrões para codificação. Com isso, o conjunto de atividades executadas por programadores recebe uma formalização, de forma que a execução de uma metodologia seja mais bem compreendida por times de desenvolvimento.

O trabalho compara uma situação inicial onde não havia um processo formal, com aquela após a implantação de um processo baseado no uso de *design patterns* em uma empresa do setor de serviços, que constrói uma série de sistemas especificamente desenvolvidos para atenderem os requisitos de seus negócios. Essa comparação está fundamentada em indicadores como tempo de codificação, incidentes ocorridos em produção e número de manutenções evolutivas e corretivas.

1.3. Motivação

O desenvolvimento de software cada vez mais enfrenta riscos conhecidos e novos como desempenho abaixo do esperado, orçamento e cronogramas pouco realistas, construção de sistemas não desejados ou simplesmente incorretos e mudanças de requerimentos pelas partes envolvidas (incluindo clientes, profissionais de tecnologia de informação e concorrência). E há uma expectativa de que as soluções sejam entregues o mais rapidamente possível (MESO, 2006).

A consequência da pressão crescente para a entrega de sistemas dentro do orçamento e prazo das organizações é o descuido com a qualidade do software. Isso ocorre quando não há um processo estabelecido para que a programação siga critérios objetivos e que sejam mensuráveis.

Entretanto, ao sacrificar a qualidade no desenvolvimento de software, organizações acabam incorrendo em custos que dificilmente são percebidos, relacionados à dificuldade em manterem-se sistemas mal escritos e documentados (ESPINDOLA & MAJDENBAUM, 2004).

O uso de *design patterns* facilita a manutenção e evolução dos sistemas. Eles permitem a estruturação dos projetos de software e facilitam a legibilidade dos códigos (ESPOSITO & SALTARELLO, 2009). A combinação desses atributos leva à redução dos custos de manutenção, por facilitar o entendimento do que um sistema deve executar.

1.4. Justificativa

A não utilização de boas práticas de programação seria decorrência da falta de um modelo estruturado de desenvolvimento, que institucionalize formas de codificação baseadas em métodos de engenharia. Como engenharia entende-se a disciplina para projetar, construir e adaptar modelos, técnicas e ferramentas para desenvolvimento (BRINKKEMPER, 1996) que neste trabalho diz respeito a softwares.

Para que a codificação de programas se alinhe com práticas de engenharia, é necessário um processo de uso das boas práticas, com a medição dos resultados obtidos. Com o tempo a programação torna-se mais previsível e há um consequente

aumento de produtividade, pois os desenvolvedores não precisam resolver problemas resolvidos por *design patterns* que já foram utilizados anteriormente.

Nos modernos ambientes de desenvolvimento integrados há ferramentas voltadas para a aplicação de boas práticas de programação, conferindo se elas realmente foram seguidas. Essas práticas contribuem para a entrega de aplicações que atendem os requisitos funcionais sem sobrecarregar os recursos da infraestrutura onde são executadas.

1.5. Metodologia

Na elaboração deste trabalho, é aplicada uma metodologia que consiste nos seguintes passos:

a) Pesquisa – Levantamento de trabalhos acadêmicos e em artigos publicados em revistas, com a seleção de itens relacionados ao objetivo do trabalho e catalogação para elaboração das referências bibliográficas. A pesquisa ocorreu ao longo de todo o trabalho, para embasar os argumentos nele contidos.
b) Análise – Escolha das informações relevantes para o trabalho nos trabalhos acadêmicos e artigos que foram pesquisados.
c) Definição do processo – Elaboração da proposta do processo que é objeto deste trabalho. Essa proposição foi construída com base na experiência profissional do autor, apoiada pelos dados coletados no passo de análise.
d) Estudo de caso – Apresentação da empresa objeto do estudo e descrição sobre como foi implantado o processo. No final são apresentados os resultados obtidos nesse estudo.

1.6. Estrutura do Trabalho

O presente trabalho está dividido em cinco capítulos, com a distribuição de conteúdo de acordo com a relação a seguir:

- Capítulo 1 – Considerações iniciais, objetivo do trabalho, motivação baseada em necessidades existentes, justificativa baseada em outros estudos e descrição da metodologia utilizada para desenvolver o trabalho.
- Capítulo 2 – Fundamentação teórica dos assuntos relacionados ao objetivo deste trabalho. Nele é detalhado o conceito de *design patterns* e como usá-los na definição de um *framework* de desenvolvimento. Também é descrito o método para elaborar o processo baseado em *frameworks* de desenvolvimento.
- Capítulo 3 – Definição do processo que visa à adoção de *framework* de desenvolvimento.
- Capítulo 4 – Explanação sobre estudo de caso da aplicação do processo para utilização de *framework* de desenvolvimento.
- Capítulo 5 – Conclusões obtidas a partir dos resultados obtidos no estudo de caso, com comentários, contribuições acadêmicas que foram obtidas e sugestões para outros trabalhos que são passíveis de desenvolvimento a partir da solução proposta.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. *Design Patterns* e *Frameworks* de Desenvolvimento

No desenvolvimento de software há questões que se repetem em construções de sistemas bastante distintos, mas as respostas para elas são análogas. A ideia de formalizar o reaproveitamento de soluções para problemas conhecidos de programação tornou-se popular graças ao sucesso do livro **Design Patterns: Elements of Reusable Object-Oriented Software** (GAMMA, HELM, JOHNSON, & VLISSIDES, 1994), dos autores que se tornaram conhecidos por *Gang of Four* ou apenas GoF.

Um *design pattern* é uma solução reutilizável de uso geral para um problema comumente encontrado em projetos de software. Ele não é uma solução pronta que pode ser transformada diretamente em código e sim uma descrição ou *template* sobre como resolver um problema, que pode ser usada nas mais diversas situações (MU & JIANG, 2011).

O uso do termo *pattern* é derivado dos trabalhos do arquiteto Christopher ALEXANDER, que embora relacionados com arquitetura e planejamento urbano se mostraram aplicáveis a inúmeras disciplinas, entre elas o desenvolvimento de software (APPLETON, 1997). Para Alexander (1979):

> Cada *pattern* é uma regra de três partes, que expressam uma relação entre certo contexto, um problema e uma solução.
>
> Como um elemento no mundo, cada *pattern* é um relacionamento entre certo contexto, certo sistema de forças que ocorre repetidamente naquele contexto e certa configuração espacial que permite a essas forças se resolverem.
>
> Como um elemento de linguagem, um padrão é uma instrução a qual mostra como a configuração espacial pode ser usada repetidamente para resolver dado sistema de forças, onde quer que o contexto a torne relevante.
>
> O *pattern* é, em resumo, ao mesmo tempo uma coisa que aconteça no mundo e a regra que nos diz como criar aquela coisa e quando precisamos cria-la. (...) É

> ao mesmo tempo a descrição da coisa que é viva e a descrição do processo que gera aquela coisa.

Para que um *design pattern* seja realmente útil na resolução de problemas recorrentes na construção de sistemas, ele deve possuir uma série de atributos, que permitam encaixá-lo dentro da regra de três partes segundo Alexander. De acordo com COPLIEN um bom *pattern*:

- *Resolve um problema*: *Patterns* capturam soluções, não apenas princípios abstratos ou estratégias.
- *É um conceito provado*: *Patterns* capturam soluções com um registro de acompanhamento, não teorias ou especulações.
- *A solução não é obvia:* Os melhores *patterns* geram uma solução para um problema indiretamente, o que é uma abordagem necessária para problemas de projeto mais difíceis.
- *O* pattern *tem um componente humano significativo* (*minimizam intervenção humana*)*:* Todos os softwares servem ao conforto humano ou qualidade de vida; os melhores *patterns* apelam de maneira explicita à estética e utilidade (COPLIEN, s/d).

O mérito da GoF foi traduzir o modelo conceitual de *patterns* em descrições de problemas comumente encontrados na construção de sistemas, agrupando-os por cenários que facilitam o entendimento pelos desenvolvedores que precisarem aplica-los. Resumidamente, os *design patterns* catalogados pela GoF são classificados em três tipos:

- Criação (*Creational Patterns*): São usados para resolução de problemas relacionados à criação de instancias de objetos.

- Estrutural (*Structural Patterns*): Definem maneiras para compor objetos de forma a obter-se novas funcionalidades destes.
- Comportamento (*Behavioral Patterns*): São usados para resolução de problemas de comunicação entre objetos.

O trabalho da GoF é baseado no uso de programação orientada a objetos (POO), modelo adotado pela maioria das linguagens de programação mais usadas pelo mercado (TIOBE Programming Community Index for January 2012, 2012). A consequência do emprego sistemático de *design patterns* em POO é o estabelecimento de *frameworks* para desenvolvimento de aplicações. De acordo com a GoF (1994):

> Um *framework* é um conjunto de classes em cooperação que fazem um *design* reutilizável para uma específica classe de software. Um *framework* provê um guia arquitetural pelo particionamento do *design* em classes abstratas e define suas responsabilidades e colaborações. Um desenvolvedor customiza um *framework* para uma aplicação em particular pela subclassificação e composição das classes do *framework*.

Outra definição de *frameworks* para desenvolvimento de aplicações é dada por APPLETON (1997):

> Um *framework* de software é uma mini arquitetura reutilizável que provê a estrutura genérica e o comportamento para uma família de abstrações de software, junto com um contexto de metáforas que especificam sua colaboração e uso dentro de um dado domínio.
>
> Um *framework* não é uma aplicação completa e sim a infraestrutura e mecanismos que executam uma política para interação entre componentes abstratos dentro de implementações abertas.

Segundo a GoF, as principais diferenças entre *design patterns* e *frameworks* são:

- *Design Patterns* são mais abstratos que *Frameworks* - *Frameworks* são incorporados ao código, mas apenas exemplos de *design patterns* são usados na codificação. Um ponto forte do uso de *framework* é ser escrito em linguagens de programação, permitindo a execução e o reuso diretamente. Já *design patterns* precisam ser implementados cada vez que forem usados. *Design patterns* também explicam a intenção, restrições e consequências do seu uso.
- *Design Patterns* são elementos arquiteturais menores que *Frameworks* - Um *Framework* típico contém vários *design patterns*, mas o reverso nunca é verdadeiro.
- *Design Patterns* são menos especializados que *Frameworks* - *Frameworks* sempre tem um domínio de aplicação específico. Em comparação, *design patterns* são usados em praticamente qualquer tipo de aplicação. Enquanto *design patterns* mais especializados são certamente possíveis, mesmo esses não estabeleceriam uma arquitetura de aplicação.

Design Patterns são empregados tanto no projeto como na documentação de *frameworks* de desenvolvimento. Um *framework* tipicamente abrange o uso de vários *design patterns*, assim, ele seria a implementação de um sistema de *design patterns*. Apesar de se relacionarem dessa forma, *frameworks* e *design patterns* são entidades distintas: um *framework* é um software executável, enquanto que *design patterns* representam conhecimento e experiência sobre software. Ou seja, *frameworks* são a realização física de um ou mais soluções e *design patterns* são as instruções de como implementar essas soluções (APPLETON, 1997).

2.2. Aplicação de *Frameworks* de Desenvolvimento

Frameworks de desenvolvimento abstraem do desenvolvedor a necessidade de conhecer *design patterns*. Além disso, como *design patterns* são soluções conhecidas para problemas comumente encontrados, o uso deles pelos *frameworks* aumenta a previsibilidade do funcionamento dos sistemas.

Para contemplar os principais requisitos técnicos na construção de sistemas, os *frameworks* de desenvolvimento são divididos em funcionalidades específicas como apresentação, segurança, etc.. A Figura 1 mostra exemplo de diagrama de classes para um *framework*.

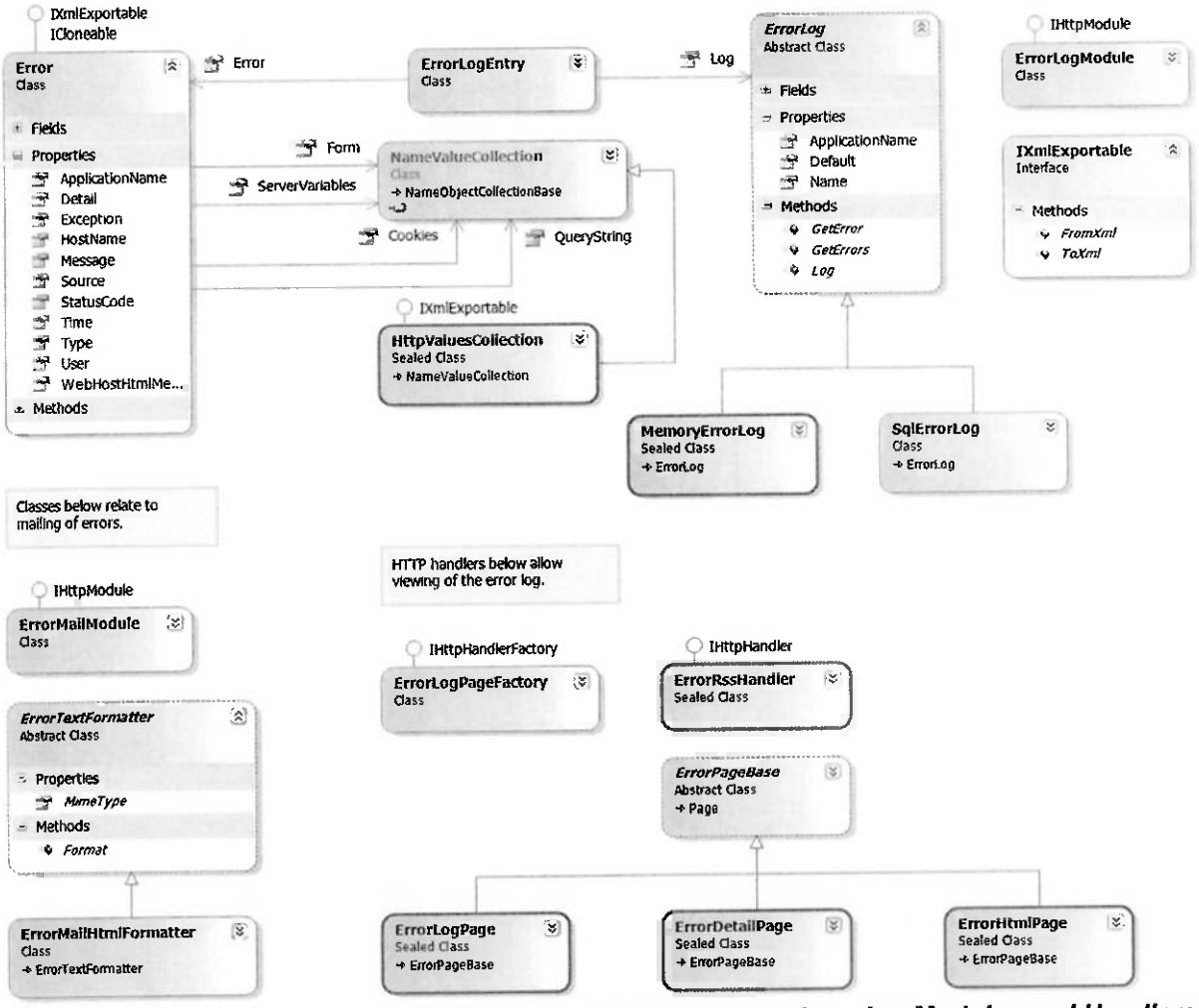

FIGURA 1 – Diagrama de Classes do *Framework* ELMAH - *Error Logging Modules and Handlers* (AZIZ, 2003-2011).

Embora o uso de *design patterns* busque melhorar a qualidade de codificação, é necessária a capacitação dos profissionais que os empregarão na construção de *frameworks* de desenvolvimento. Se não houver entendimento claro de como utilizá-los, pode-se ter a consequência indesejável de impactar negativamente a qualidade da construção de sistemas. Em estudo conduzido por KHOMH & GUÉHÉNEUC, observou-se que o uso de alguns dos *design patterns* listados pela GoF atingiu o resultado oposto ao esperado em aspectos como reuso, expansibilidade e entendimento do código (KHOMH & GUÉHÉNEUC, 2008).

A vantagem dos *frameworks* de desenvolvimento para as organizações está na produtividade decorrente do reuso. Um *framework* de desenvolvimento é um conjunto de rotinas aplicáveis no desenvolvimento de diferentes aplicações em uma organização. Com isso, desenvolvedores não iniciariam a codificação do zero. O *framework* proveria a maioria das partes necessárias que então seriam customizadas e conectadas para formar essas aplicações (CWALYNA & ABRAMS, 2009, p. 2).

2.3. Adoção de *Frameworks* de Desenvolvimento

Os *frameworks* de desenvolvimento construídos com base em *design patterns* remetem ao uso de metodologia orientada a objeto (MOO). As MOOs têm a intenção de impulsionar a construção de software, auxiliar o gerenciamento e produtividade do processo de desenvolvimento, melhorar a qualidade dos sistemas e facilitar o reuso de código. Entretanto pesquisa feita por WOO, MIKUSSAUKAS, BARLETT & LAW mostra que apenas 42,7% das organizações que fizeram parte do escopo afirmaram que tinham adotadas MOOs até determinado ponto, conforme Tabela 1.

TABELA 1 - Adoção de MOO por tipo de negócio (WOO, MIKUSAUKAS, BARLETT, & LAW, 2006).

Tipo de Negócio	Status de adoção de MOO		
	Sem planos de uso	Plano para uso em até três	Está usando
Educação	2	4	2
Bancos/Finanças/Seguros	1	2	7
Agricultura/Mineração/Petróleo	2	0	1
Governo	6	4	3
Manufatura/Engenharia	1	2	3
Médico/Hospitalar	1	2	4
Comércio/Marketing	1	1	1
Desenvolvimento de Software	2	4	6
Telecomunicações	2	2	4
Transporte	3	0	2
Utilidades	1	0	2
Consultoria/Gerenciamento/Serviços	2	1	0
Entretenimento/Jogos	0	1	0
TOTAL	***24***	***23***	***35***

Sem o uso de uma MOO, torna-se mais difícil atingir um dos principais resultados na adoção de um processo de construção de sistemas utilizando *frameworks* de desenvolvimento, que é o reuso de código. Afinal, nem mesmo o uso de orientação a objetos garante a criação de objetos reusáveis, pois depende do analista de sistemas planejar o reuso quando conduz a análise orientada a objeto (SHERIF & VINZE, 2003).

Este trabalho não diz respeito à adoção de metodologia de desenvolvimento. Porém o processo de uso de *frameworks* de desenvolvimento endereça questões relativas ao uso de padrões, algo necessário para estabelecimento de um modelo bem definido para construção de sistemas de informação. Logo o processo torna-se um dos elementos de apoio para suportar a implantação de uma metodologia.

2.4. Processo de Uso de *Frameworks* de Desenvolvimento

Com um processo baseado em *frameworks* de desenvolvimento para construção de sistemas, as organizações maximizam os benefícios oferecidos pelas modernas

plataformas de desenvolvimento em termos de produtividade na codificação e qualidade do produto final, principalmente pelo uso correto do reuso de código. Além disso, o conhecimento de como foram construídos os sistemas torna-se algo coletivo, evitando situações como os sistemas com donos e a dificuldade de integrar novos membros aos times de desenvolvimento pela falta de uma documentação adequada da execução das atividades de codificação.

Para estabelecer o uso de *frameworks* de desenvolvimento, este trabalho utiliza o método de instanciação de processos segundo Panigassi (2007) e Dias (2010). Com isso espera-se que o uso desses *frameworks* torne-se uma prática sistematizada, minimizando riscos como a resistência dos desenvolvedores em seguir padrões de codificação e a pouca importância dada por líderes de desenvolvimento quanto aos aspectos de qualidade promovidos pelos *frameworks*.

A dificuldade em instanciar um processo baseado no uso de *framework* de desenvolvimento está bastante relacionada com a flexibilidade inerente ao projeto de um *framework*. Essa flexibilidade faz com que os *frameworks* sejam complicados de entender e dificulta a tarefa de instanciação. Outro problema é a ausência de documentação de alto nível que suporte a execução do processo. Em geral, quanto mais pobre a documentação, mais difícil é o processo de instanciação. No caso, entende-se por uma documentação pobre um conjunto de modelos que expõe detalhes desnecessários do projeto e codificação internos do *framework*. Uma documentação de alto nível seria aquela que exibe características relevantes da solução que o *framework* oferece de forma a oferecer incrementos que sejam relevantes para serem adaptados por um não especialista em um dado domínio de aplicação (FILHO, OLIVEIRA, & LUCENA, 2004).

A arquitetura do processo de uso de *frameworks* de desenvolvimento segue a definição de Borsoi, pois o cenário da sua aplicação está de acordo com o definido para uma fábrica de software (BORSOI, 2008). Os modelos de representação da arquitetura de processo para uso de *frameworks* de desenvolvimento são apresentados na Tabela 2.

TABELA 2 – Relacionamento entre modelos e suas representações, classificados segundo Borsoi (2008).

Modelos	Diagrama	Descrição	Representação
Estrutural	Relação dos requisitos técnicos	Identifica os requisitos técnicos para um *framework* de desenvolvimento	Texto
	Categorização de requisitos	Identifica as categorias dos requisitos técnicos, segundo as diferentes visões de TI	Texto
	Identificação dos *design patterns*	Identifica quais são os *design patterns* que atendem os requisitos técnicos	Texto
Comportamental	Diagrama padrão de fluxo de sequência e de mensagens entre atividades	Modelo que apresenta o fluxo de artefatos e a troca de mensagens entre as atividades de um processo ou entre processos.	BPMN
	Diagrama padrão de camadas	Modelo que apresenta a divisão por camadas das funcionalidades do *framework* de desenvolvimento	Diagrama de Camadas
	Diagrama padrão de classes	Modelo que apresenta as classes que compõe o *framework* e identifica os *design patterns* adotados.	Diagrama de Classes UML

O desenho do processo de uso de *frameworks* de desenvolvimento é feito segundo a notação BPMN. Essa é uma notação de uso disseminado no mercado e suportada por várias ferramentas como o Microsoft Visio e até mesmo por software livre, por exemplo o ProcessMaker (www.processmaker.com).

3. DEFINIÇÃO DO PROCESSO

O processo para construção de sistemas baseado em *frameworks* de desenvolvimento não contempla todas as necessidades de uma metodologia orientada a objetos para desenvolvimento de software e sim os aspectos mais práticos dessa metodologia, relacionados à programação. Esse processo abstrai o uso dos *design patterns* empregados nesses *frameworks* por uma equipe de desenvolvedores, promovendo de forma sistemática a padronização da codificação e o reuso de código.

3.1. Consideração para Adoção do Processo

A demanda para criar um padrão de desenvolvimento é decorrência da evolução tecnológica da plataforma que ele atenderá. Novas tecnologias como processadores *multicore*, virtualização de servidores e computação em nuvem são mais facilmente adotadas por empresas que usam *frameworks* baseados em *design patterns*, pois estes possuem a qualidade de serem extensíveis, i.e., poderem ser usados em cenários não existentes quando da sua criação.

Os *frameworks* de desenvolvimento não se aplicam de forma imediata à construção de aplicações nas organizações. Para que sejam realmente úteis é necessário chegar aos *design patterns* por meio da refatoração de código e generalização dos problemas que se encontram na construção de aplicações (MILLETT, 2010).

Para que os padrões sejam efetivamente seguidos, a organização precisa deliberar os papéis correspondentes à definição, divulgação e aperfeiçoamento deles. Caso se opte por criar internamente um *framework* de desenvolvimento, também será necessário o papel do construtor, que deverá conhecer não só a plataforma onde

será usado o *framework*, mas igualmente precisará dominar o uso de *design patterns* como forma para incrementar o reuso de código.

A escolha entre as possíveis alternativas de adoção de um padrão de desenvolvimento não é simples. Por envolver aspectos qualitativos (por exemplo, experiência técnica de profissionais), a contratação de uma consultoria especializada pode ajudar na decisão. Há diferentes empresas que oferecem esse serviço em função da plataforma de desenvolvimento a qual o padrão se aplicará. E se houver interesse na construção de um padrão próprio, pode-se fazer um *benchmark* com outras organizações que seguiram essa direção.

A adoção de um *framework* de desenvolvimento pode ser pela adoção de um oferecido pelo mercado (por exemplo, *Java 2 Enterprise Edition*, *Microsoft Enterprise Library*) ou por meio da criação de um específico. O uso de um *framework* de mercado é viável quando ele resolve de forma adequada os requisitos técnicos mais comumente encontrados na construção de aplicações de uma organização. Já a criação de um *framework* próprio requer desenvolvedores com a capacidade de entender os aspectos relacionados à aplicação de *design patterns* para reuso de rotinas entre diferentes aplicações.

Embora um bom *framework* necessite um nível de abstração que facilite a compreensão de como empregá-lo, é necessário estabelecer as regras para seu uso via algum mecanismo formal. Por exemplo, um problema típico encontrado na construção de sistemas é a falta da documentação correspondente e o mesmo vale para os padrões usados no desenvolvimento deles. Outras questões como a falta de conhecimento de técnicas de documentação ou a inexistência de um repositório

adequado para os artefatos produzidos, leva à progressiva perda do reuso de funcionalidades já utilizadas anteriormente.

Os padrões de desenvolvimento devem ter peso de norma para serem usados de forma ampla. Sem isso, as equipes de desenvolvimento não têm o compromisso em atendê-los. O risco é que normas tendem a ser percebido como algo imposto, o que eventualmente leva ao surgimento de resistências, especialmente quando há equipes com desenvolvedores que tenham a impressão de serem tão qualificados quanto os profissionais que definem os padrões.

Uma implantação bem sucedida do processo de uso de *frameworks* de desenvolvimento depende de um patrocinador forte, que normalmente é o diretor responsável pela área de desenvolvimento. Ele deve reconhecer a importância do investimento em melhoria da qualidade de software, pois algumas das vantagens em se ter um *framework* aparecem somente em médio prazo, como a diminuição de custo em manutenções corretivas.

A avaliação dos resultados obtidos com a adoção de padrões se dá pela verificação do tempo de desenvolvimento de aplicações, onde é esperada diminuição progressiva à medida que esses padrões sejam amadurecidos. Outro benefício esperado é a redução de incidentes relacionados ao mau funcionamento de aplicações, considerando que o não uso de um *framework* de desenvolvimento leva à construção de sistemas que apresentam um número alto de incidentes em produção, com o consequente retrabalho para ajustá-los.

3.2. Modelagem do Processo

Com base no trabalho de Panigassi (2007), a modelagem do processo para construção de sistemas baseado em *frameworks* de desenvolvimento, parte da definição dos requisitos de processo. No entanto, modelos como CMMI tratam requisitos de negócio, enquanto *frameworks* de desenvolvimento buscam resolver problemas ligados aos requisitos técnicos ligados à construção de sistemas.

A proposta do modelo conceitual de um ciclo de vida para o processo visando estabelecer o uso de *frameworks* de desenvolvimento, representado na Figura 2, compreende as seguintes etapas:

- Elaboração dos objetivos do *framework* de desenvolvimento – etapa onde os requisitos em comum dos sistemas – acesso a dados, segurança, monitoramento, interface do usuário, etc. – são identificados e se estabelece a estratégia para adoção de um *framework* de desenvolvimento.
- Especificação do processo de uso do *framework* de desenvolvimento – etapa onde são categorizados os requisitos que farão parte do *framework* de desenvolvimento e é feita a escolha dos *design patterns* adequados.
- Construção do *framework* – etapa na qual o *framework* é construído com base na escolha de *design patterns* efetuada na etapa anterior.
- Implantação do processo de uso do *framework* – etapa onde o *framework* é validado em pilotos e posteriormente liberado para uso pelas equipes de desenvolvimento, com acompanhamento dos resultados obtidos.

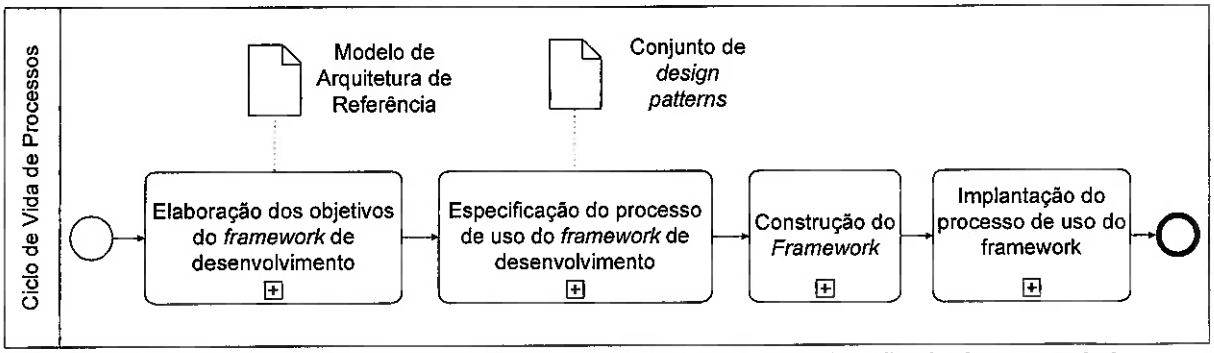

FIGURA 2– Etapas do modelo conceitual para ciclo de vida de definição de *framework* de desenvolvimento, adaptado de Panigassi (2007).

A arquitetura do processo para uso de *frameworks* de desenvolvimento também parte de uma arquitetura de referência, conforme proposto por Dias (2010). Essa arquitetura é definida com base nos requisitos técnicos mais comumente encontrados nos diferentes sistemas mantidos por uma equipe de desenvolvedores, em conjunto com os recursos computacionais à disposição (sistemas operacionais, bancos de dados, serviços de *middleware*, etc.). A arquitetura de referência não é gerada a partir de modelos de qualidade como CMMI, MPS.BR ou ISO 12207, pois neste caso ela lida com aspectos técnicos da plataforma de desenvolvimento onde será aplicada (Java, .NET Framework etc.), mas oferece suporte para adoção desses modelos.

Um *framework* de desenvolvimento para ser bem sucedido deve ser aplicável a qualquer cenário de desenvolvimento, independente dos requisitos de negócio a serem atendidos pelos sistemas em construção. Portanto, a partir da arquitetura de referência parte-se diretamente para a arquitetura do projeto, onde serão identificadas quais funcionalidades previstas pelo *framework* atendem os requisitos técnicos derivados daqueles de negócio.

Embora o processo para utilização de *framework* de desenvolvimento não contemple a arquitetura operacional, é necessário considerar que ele é pensado para constituir

uma das partes de um processo mais amplo de implantação de uma metodologia de desenvolvimento de sistemas. Ou seja, visto de uma perspectiva abrangente, a arquitetura de referência para uso de *framework* de desenvolvimento é uma arquitetura operacional para um modelo de qualidade voltado à construção de sistemas.

3.2.1. Elaboração dos Objetivos

Qualquer que seja o padrão de desenvolvimento, para que tenha sucesso é fundamental que atenda às necessidades do negócio da organização. Por exemplo, uma empresa de comércio eletrônico necessita de sistemas que suportem um grande número de transações simultâneas de atualização de dados, delegando para uma solução de *business intelligence* a consolidação dos dados recebidos em relatórios que descrevam como é o perfil de seus clientes a partir dos seus hábitos de compra. A Figura 3 monstra as tarefas correspondentes a etapa de elaboração dos objetivos do *framework* de desenvolvimento.

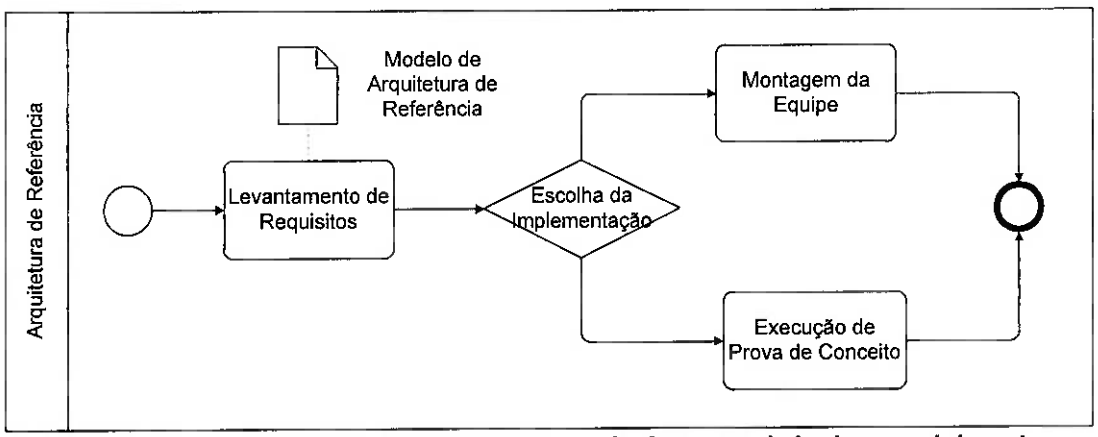

FIGURA 3 – Etapa de elaboração dos objetivos do *framework* de desenvolvimento.

A arquitetura de referência para uso de *framework* de desenvolvimento é baseada nos recursos computacionais à disposição da organização, Uma lista não restrita inclui: tipos de servidor, meios de armazenamento, topologia de redes, SGBDs, *application servers*, plataformas de desenvolvimento, entre outros. Em organizações

mais complexas uma área de Arquitetura Corporativa identifica e mantém o catálogo desses recursos.

O levantamento de requisitos para um *framework* acontece por meio do levantamento das características mais comumente encontradas nas aplicações, que indicam onde o reuso de código trará maiores vantagens. Também é necessário considerar as características da infraestrutura onde são executadas as aplicações. Ela pode dar ao construtor do *framework* recursos que proporcionam maior flexibilidade para atingir os resultados esperados.

O uso de um *framework* existente é uma opção para organizações que não tenham os recursos necessários para construir e manter um desenvolvido internamente. Particularmente na plataforma Java é uma prática corrente o uso de *frameworks* prontos, sendo que trabalhos como o de Chang et al (2011) mostram a viabilidade do uso deles por equipes de desenvolvimento voltadas ao uso de padrões. Do lado da plataforma .NET Framework, iniciativas como o CodePlex (www.codeplex.com) oferecem um conjunto de *frameworks* aplicáveis em diversos cenários.

Ao decidir pelo uso de um *framework* de desenvolvimento trazido do mercado, pode-se fazer uma avaliação para escolha daquele que melhor atende às necessidades da organização. Para auxiliar na tomada de decisão podem-se usar técnicas como a execução de provas de conceito, onde os *frameworks* são aplicados em condições típicas da construção de sistemas.

Caso a decisão seja por se ter *frameworks* próprios, deve-se atribuir a uma equipe específica a tarefa de construí-los. Não existe uma regra para definir em que parte do cronograma da área de TI essa equipe ficará ou os critérios para escolha de seus

integrantes. O mais realista é considerar que atributos como capacitação técnica, raciocínio abstrato e conhecimento do ambiente de TI são imprescindíveis para aqueles que farão parte dessa equipe.

3.2.2. Especificação do Processo

A Figura 4 monstra as tarefas correspondentes à etapa de especificação do processo de uso do *framework* de desenvolvimento.

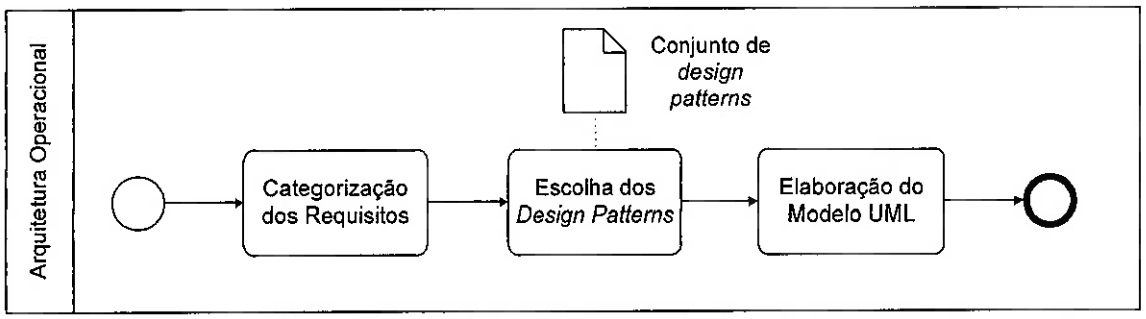

FIGURA 4 – Etapa de especificação do processo de uso do *framework* de desenvolvimento.

Na especificação do processo, as funcionalidades previstas para o *framework* de desenvolvimento são categorizadas de acordo com requisitos técnicos que irão atender. Essa técnica visa simplificar o emprego do *framework*, permitindo que somente as partes realmente necessárias para a construção de um sistema sejam empregadas. Adicionalmente é possível realizar a implementação do processo em fase, à medida que novas funcionalidades sejam agregadas ao *framework.*

O estabelecimento das funcionalidades de um *framework* acontece por meio da identificação das características mais comumente encontradas nas aplicações, que indicam onde o reuso de código trará maiores vantagens. Também são consideradas as características da infraestrutura onde são executadas as aplicações. Estes possuem recursos que dão ao construtor do *framework* maior flexibilidade para conseguir os resultados esperados.

Uma vez identificados quais funcionalidades são relevantes para o *framework*, é feita a escolha dos *design patterns* necessários para sua construção. Essa escolha se dá pela comparação entre um cenário de reuso desejado (por exemplo, persistência) com a descrição do uso de um ou mais *patterns* em busca daqueles que promovam uma solução adequada para a maioria dos usos que se espera do *framework*.

Eventualmente uma organização pode ter alguma reuso de código, feito de maneira empírica. Dessa forma, a construção de um *framework* pode começar pela identificação de componentes que tenham características adequadas para adoção em diversos projetos de desenvolvimento, principalmente pelo uso de *design patterns* apropriados.

Após a escolha dos *design patterns* a serem empregados na construção dos *frameworks* de desenvolvimento, elabora-se o diagrama de classes UML para os componentes que irão compor esses *frameworks*. Nesta tarefa confirmam-se as escolhas feitas e cria-se uma representação a ser usada na etapa de construção do processo, quando são codificados os componentes.

3.2.3. Construção do Framework

A escolha entre um método mais tradicional, baseado em práticas do CMMI, ou ágil usando técnicas como Scrum ou XP é irrelevante para a forma como *frameworks* são construídos. Deve-se considerar que a adoção de *frameworks* deve ser agnóstica em relação ao modelo de ciclo de vida onde for aplicado o uso de *frameworks* de desenvolvimento. Na Figura 5 são apresentadas as tarefas correspondentes à etapa de especificação do processo de uso do *framework* de desenvolvimento.

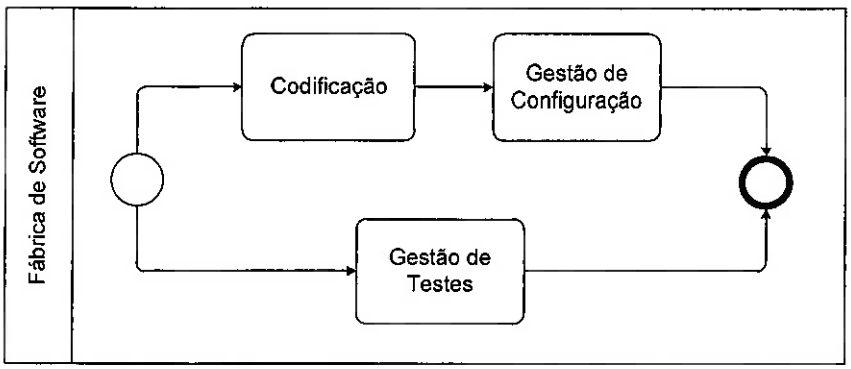

FIGURA 5 – Etapa de construção do *framework* de desenvolvimento.

A realização do *framework* enquanto produto final se dá por componentes de código. Estes são identificados por termos como API, bibliotecas, componentes, etc. Como todo produto, é necessário fabricá-los e isso se faz por codificação, de forma similar à construção de sistemas. A diferença está no fato de que a programação não está voltada á resolução de problemas de negócio, mas para abstrair questões técnicas que podem ocorrer em qualquer construção de sistemas.

Os artefatos produzidos na construção de *frameworks* de desenvolvimento devem atender os procedimentos de gestão de configuração em uso pela organização. Particularmente importante é o versionamento, que consiste em rotular os componentes produzidos com um número de versão, relacionando as suas funcionalidades e eventualmente listar quais foram os *bugs* encontrados, para referência futura. No mercado há diversas ferramentas que contemplam as necessidades de um processo de gestão de configuração controlado.

Desde o início da codificação devem-se prever os testes para os componentes dos *frameworks* de desenvolvimento. Neste trabalho não é detalhada a realização desses testes, porém recomenda-se o uso de técnicas de TDD, como testes unitários e integrados, na construção dos *frameworks.* O importante é garantir que nenhuma falha grave de codificação seja descoberta em *framework* de desenvolvimento durante seu uso na construção de aplicações.

3.2.4. Implantação do Processo

A Figura 6 monstra as tarefas correspondentes à etapa de implantação do processo de uso do *framework* de desenvolvimento.

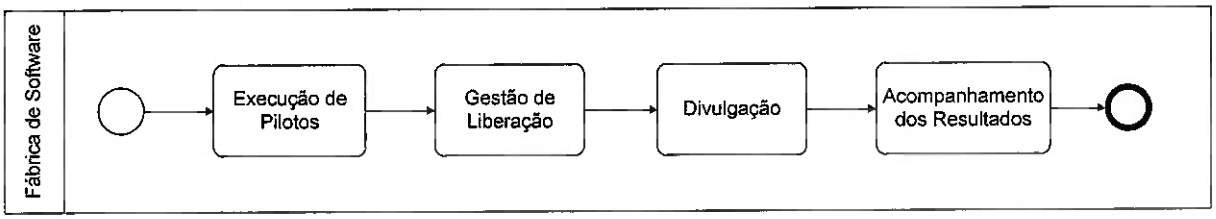

FIGURA 6 – Etapa de implantação do processo de uso do *framework* de desenvolvimento.

O uso de *frameworks* de desenvolvimento começa na execução de pilotos, no caso a construção de aplicações que promovam o uso desses *frameworks*. As escolhas de quais desenvolvimentos farão parte do piloto pode acontecer em qualquer etapa do ciclo de vida para definição do *framework*, estando sujeita às necessidades da organização.

A gestão de liberação dos componentes que fazem parte dos *frameworks* de desenvolvimento precisa considerar o fato de que eles serão usados por diferentes aplicações. A recomendação neste caso é limitar a dependência entre a implantação desses componentes e as aplicações que o utilizarão. Uma boa prática é distribuir os componentes dos *frameworks* antes das aplicações que os utilizam. Isso permite antecipar e corrigir eventuais problemas de forma a minimizar o risco de prejudicar o funcionamento dos sistemas dependentes desses *frameworks*.

Para que os desenvolvedores de uma organização tomem conhecimento dos *frameworks* e quando aplicá-los é necessário um plano de comunicação adequado às características do público-alvo que irá utilizá-los. Esse plano depende de fatores como tamanho da equipe de desenvolvimento, idade média dos profissionais, entre outros.

O surgimento de plataformas colaborativas para as intranets das organizações permite atacar tanto o problema de divulgação dos padrões quanto o de aceitação deles. Fóruns de discussão criam debates sobre a melhoria dos padrões, com a participação de desenvolvedores com diferentes visões e necessidades. *Wikis* criam uma base de conhecimento que preserva a experiência acumulada na construção de sistemas. Por sua vez, recursos como *workflows* de aprovação dão à documentação dos padrões a relevância para que estes sejam entendidos como algo imprescindível para a construção de aplicações. Uma objeção quanto a confiar em portais é que nem todos possuem o hábito de lê-los.

A documentação dos padrões não deixa de ser um artefato de desenvolvimento e, portanto, necessita estar sob o controle de algum processo formal de gestão de configuração. Assim são usados mecanismos de versionamento, que permitem controlar as mudanças ocorridas e medir a velocidade com que ocorrem.

O treinamento para utilização de um *framework* de desenvolvimento, por sua vez, é um mecanismo eficaz quanto à capacitação dos profissionais envolvidos. Nele é possível tirar dúvidas e colocar em prática as lições aprendidas, para fixação dos conceitos. A questão é que esses treinamentos têm um custo relativamente alto, pois se considerar o tempo de cada participante (incluindo o instrutor) pode-se chegar numa quantidade de horas/homem difícil de justificar para aqueles que pagam os custos de desenvolvimento. E à medida que os padrões evoluam, faz- se necessária sua reaplicação, para incorporar as novidades desde a última sessão.

Uma combinação entre treinamento e plataformas colaborativas está na publicação de treinamentos *online* que reproduzem a preleção que seria dada em uma sala de aula. Esses treinamentos podem dar vários benefícios, como acompanhamento de

acordo com a disponibilidade dos participantes, a liberação dos recursos de sala de aula e a possibilidade de serem aplicados em novos membros do time de desenvolvimento.

O acompanhamento dos resultados se dá entre a comparação dos resultados obtidos após a adoção do processo com dados históricos que a organização possua. Um particularmente adequado para comprovar o sucesso na adoção de *framework* de desenvolvimento é o número de linhas de código, informação facilmente obtida a partir de IDEs de desenvolvimento e ferramentas de QA. Outra medida da eficiência do processo é o tempo de desenvolvimento, extraído a partir de ferramentas de gestão de projetos. E ao entrar em produção, espera-se que as aplicações construídas com o uso do processo baseado em *framework* de desenvolvimento apresentem menos problemas em produção.

Em resumo, os dados quantitativos para medir os resultados do processo normalmente apresentarão valores menores após a adoção de *frameworks* de desenvolvimento. Eventualmente uma organização pode estar num estágio onde não tenha medidas relacionadas ao desenvolvimento de sistemas. Nesse caso a adoção do processo seja justamente uma das tarefas para realizar um maior controle na construção de aplicações.

4. ESTUDO DE CASO

Este capítulo do trabalho apresenta uma situação real, encontrada em uma empresa de grande porte do setor de serviços e para esse contexto foram seguidas as quatro etapas para criação de um *framework* apresentadas no capítulo anterior. Os resultados são analisados pela comparação entre a situação antes e depois da implantação do processo.

4.1. Definição do Estudo de Caso

O domínio para a aplicação de um processo para utilização de *frameworks* de desenvolvimento foi uma empresa de grande porte do setor de serviços. Seu diferencial de negócio está na sua base de dados, a maior do mercado onde atua e utilizada por sistemas de informação capazes de processar um grande volume de transações.

A motivação para o estudo de caso é definir uma forma para que aplicações tenham um alto nível de qualidade, difícil de atingir sem o uso de técnicas adequadas, e sejam entregues dentro dos prazos que os clientes internos e externos exigem.

A criação do processo de uso de *frameworks* de desenvolvimento é apresentada sob a perspectiva dos especialistas de TI, responsáveis pela construção dos componentes que integram esses *frameworks*. Também foi considerada a participação das equipes de desenvolvimento, responsáveis por praticar o uso de *frameworks* na construção de sistemas de informação.

4.2. Cenário Inicial

O desenvolvimento de sistemas na empresa foco deste trabalho começou em uma plataforma *mainframe*, utilizando técnicas de análise estruturada para construção das aplicações. Ocorreu uma evolução significativa quando as informações passaram a ser armazenadas em SGBD no lugar de arquivos ISAM (*Indexed Sequencial Access Method*), pois houve preocupação em garantir que os novos recursos fossem utilizados da melhor forma possível. Essa foi a primeira prática visando qualidade de software dentro da empresa.

A popularização dos computadores pessoais e a consequente redução de custos, fez com que a empresa começasse a disponibilizar *desktops* para seus funcionários. Em adição, começou o desenvolvimento de sistemas administrativos, voltados não para os sistemas-chave do negócio e sim para tarefas operacionais. A construção desses novos sistemas se deu em bases departamentais, sem apoio de um modelo de qualidade comum para construção deles.

O advento da internet como novo canal para disponibilizar as informações processadas pela empresa trouxe o desafio de integrar os sistemas rodando no *mainframe* com interfaces do usuário baseadas no uso de navegadores. A plataforma escolhida para construção de aplicações web, adequadas ao novo cenário, foi a Java. Foi necessária a contratação de novos desenvolvedores, com experiência na linguagem e conhecimento de análise orientada a objetos.

A empresa se deparou com a falta de profissionais para completar o quadro de desenvolvedores que construiriam as aplicações web. No início do século XXI estava em plena execução o projeto SPB (Sistema de Pagamentos Brasileiro) do Banco Central e as instituições financeiras demandavam um grande número de

programadores Java, tendo em vista que essa plataforma era a que melhor atendia os requisitos técnicos do SPB. A consequência para a organização foi a adoção de plataforma da Microsoft para desenvolvimento da intranet, uma nova demanda que surgiu em consequência do aumento da complexidade dos processos internos.

A necessidade de se ter rapidamente aplicações web, devido à demanda dos clientes internos e externos por esse tipo de software, fez com que a qualidade dos sistemas de informação fosse visto como algo secundário. O impacto dessa escolha foi um número crescente de incidentes em produção, relacionados ao baixo desempenho das aplicações. Como elas não foram construídas de forma escalar, ocorriam problemas quando o volume de transações ultrapassava determinados limites.

Em meados da década passada a empresa atentou para o fato de que estava com três plataformas distintas, sem nenhum ponto em comum entre elas (a não ser integração dos dados) e com as aplicações de internet e intranet apresentando graves falhas de funcionamento em decorrência da falta de padrões de desenvolvimento. Uma possível justificativa para essa seria o fato de que as ferramentas IDE usadas no desenvolvimento de aplicações web e departamentais aumentaram muito a produtividade dos desenvolvedores. Com isso, eles conseguiram atender rapidamente os requisitos de negócio, sem considerar outros aspectos da construção de sistemas, como desempenho e reuso de código.

Com o tempo observou-se que algumas das aplicações comportavam-se de forma adequada. A diferença em relação às que apresentavam problemas é que elas foram construídas utilizando componentes reusáveis, mesmo sem o apoio de um processo formal para adotá-los. Essa constatação levou à formação de uma equipe

para manutenção desses componentes, constituída pelos desenvolvedores que os codificaram.

A identificação de componentes reusáveis não levou de imediato ao estabelecimento de padrões de desenvolvimento. Como não foi definido um processo para uso desses padrões, a maioria das equipes de desenvolvimento não tomou conhecimento deles.

Há cerca de cinco anos atrás a empresa foi adquirida por um grupo estrangeiro e houve um salto em suas operações, refletido no aumento do quadro de TI que hoje possui em torno de quinhentos funcionários. O aumento da complexidade do negócio da empresa e a dinâmica do segmento onde atua criou a oportunidade para se criar uma área de Governança de TI, que tem como uma de suas metas a institucionalização de uma metodologia de desenvolvimento de sistemas. Com isso foi identificada a oportunidade de formalizar o uso de padrões por todos os times de desenvolvimento. A intenção seria reduzir a disparidade entre entregar no prazo versus qualidade das aplicações.

A Figura 7 mostra a estrutura organizacional da empresa foco deste estudo de caso, com as áreas participantes do processo. Ele mostra que há uma distinção entre a equipe responsável pelos padrões (Suporte ao Desenvolvimento) e as demais que usarão os mesmos, de forma a minimizar eventuais conflitos de interesse na adoção do processo descrito neste trabalho.

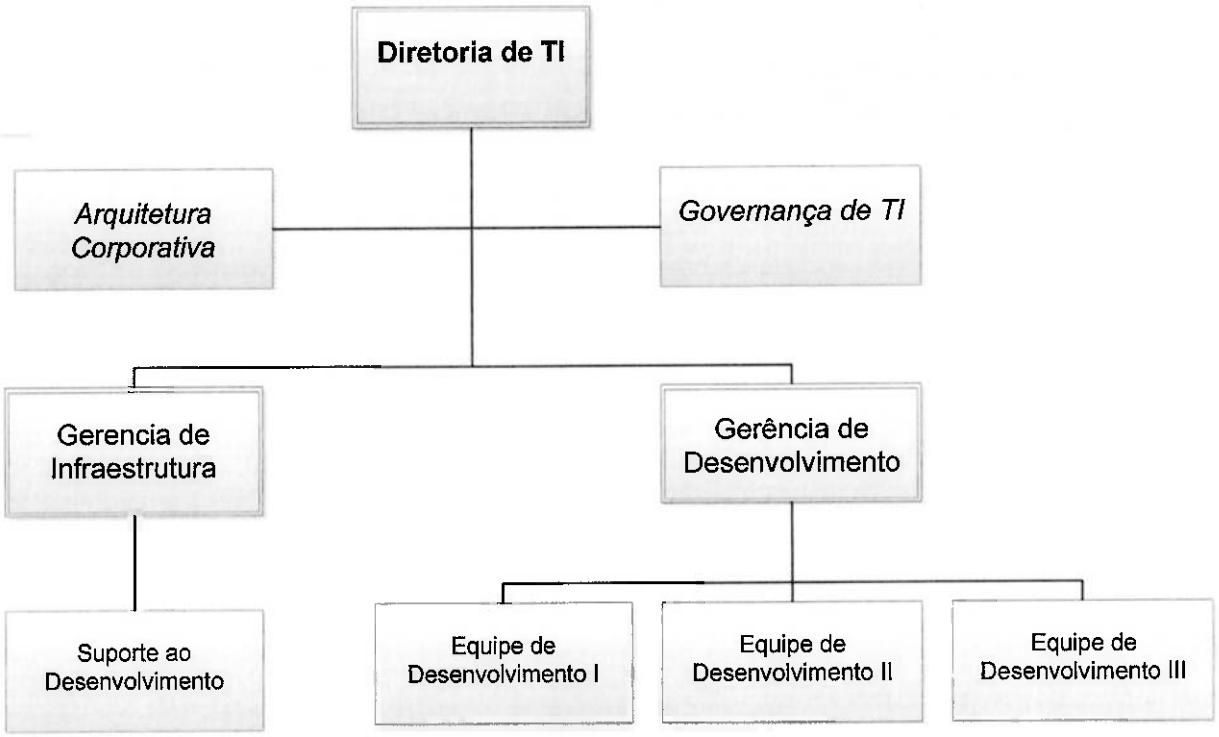

FIGURA 7 – Organograma simplificado com as áreas participantes do estudo de caso.

4.3. Planejamento do Estudo de Caso

Este estudo de caso propõe a adoção das técnicas de instanciação de processos segundo Panigassi (2007) e Dias (2010). O método de instanciação adotado é do tipo I, pois é baseado no uso de uma arquitetura de referência para a geração da arquitetura operacional. Essa escolha foi decorrente da proposta de que a construção de sistemas baseados em *frameworks* de desenvolvimento seja um padrão corporativo, com uso mandatório pelas equipes de desenvolvimento.

A realização do processo para construção de sistemas baseado em *frameworks* de desenvolvimento se deu pela criação de um projeto específico, segundo as regras de gerenciamento de projetos adotadas pela empresa. A técnica usada na gestão do projeto não será discutida neste trabalho, mas ela abrangeu a criação de atividades correspondentes a tarefas que fazem parte das etapas do ciclo de vida para definir os *frameworks* de desenvolvimento, conforme Tabela 3.

TABELA 3– Cronograma das atividades para criação do processo de uso de *frameworks* de desenvolvimento.

Nome da tarefa	Duração	Início	Término
Elaboração de Objetivos	**4 dias**	**Seg 31/10/2011**	**Sex 04/11/2011**
Levantamento de Requisitos	4 dias	Seg 31/10/2011	Sex 04/11/2011
Especificação do Processo	**5 dias**	**Seg 07/11/2011**	**Sex 11/11/2011**
Categorização dos Requisitos	2 dias	Seg 07/11/2011	Ter 08/11/2011
Escolha dos *design patterns*	2 dias	Qua 09/11/2011	Qui 10/11/2011
Elaboração do modelo UML	1 dia	Sex 11/11/2011	Sex 11/11/2011
Construção dos *Frameworks*	**29 dias**	**Seg 14/11/2011**	**Sex 23/12/2011**
Codificação	24 dias	Seg 14/11/2011	Sex 16/12/2011
Elaboração do Diagrama de Classes	5 dias	Seg 19/12/2011	Qua 16/11/2011
Gestão de Testes	29 dias	Seg 14/11/2011	Sex 23/12/2011
Implantação do Processo	**24 dias**	**Seg 02/01/2012**	**Sex 03/02/2012**
Execução dos Pilotos	17 dias	Seg 02/01/2012	Ter 24/01/2012
Gestão de Liberação	2 dias	Qui 26/01/2012	Sex 27/01/2012
Divulgação	5 dias	Seg 30/01/2012	Sex 03/02/2012

A área de Arquitetura Corporativa patrocinou a criação do processo baseado no uso de *frameworks* de desenvolvimento. Anteriormente ela já tinha estabelecido grupos de usuários das plataformas Java e .NET Framework e estes trouxeram as necessidades das equipes de desenvolvimento quanto ao uso de padrões na construção de sistemas.

Do ponto de vista tecnológico, foi adotada a estratégia de ter *frameworks* por linguagem, para se aproveitar ao máximo os recursos de cada plataforma. Foram levados em conta os recursos oferecidos pelas IDEs de cada uma das plataformas usadas (Eclipse para Java e Visual Studio para .NET Framework), por meio de customizações para que o uso dos *frameworks* fosse integrados às ferramentas. Dessa forma, este estudo de caso refere-se a *frameworks* no plural para deixar clara a intenção de que cada plataforma de desenvolvimento teria um *framework* correspondente.

4.2.1. Elaboração dos Objetivos

Na visão da empresa, a construção dos *frameworks* foi alinhada com a estratégia de implantação de uma metodologia de desenvolvimento de sistemas. Foram seguidas as orientações ditadas pela Política de Desenvolvimento Seguro, cujo objetivo é garantir que os sistemas de informação da empresa respeitem os princípios de confidencialidade, integridade e disponibilidade dos dados.

Já existia um modelo de arquitetura de referência, elaborado pelo Suporte ao Desenvolvimento. Ele foi constituído a partir do conhecimento acumulado pela equipe em acompanhar o funcionamento da infraestrutura onde as aplicações são executadas. Dados coletados por meio de monitoramento permitiram estabelecer de forma clara o que funcionava bem em oposição ao que comprometia os recursos existentes.

Aspectos técnicos, como maior volume de consultas que atualizações nas transações *online* e predominância de processamento em lote (*batch*) para atualização das bases de dados, foram considerados nesta etapa. E ainda uma série de requisitos legais que os sistemas de informação da empresa devem respeitar influenciaram as escolhas feitas durante a elaboração dos objetivos para os *frameworks* de desenvolvimento.

Sob a visão de engenharia, era necessário que os *frameworks* de desenvolvimento simplificassem para os desenvolvedores o atendimento de certos requisitos técnicos que tendem a uma codificação prolixa e com baixo reuso de código. Um exemplo é o acesso a informações em bancos de dados relacionais, onde a execução de uma *query* montada em tempo de execução gera um processamento muito maior do que

a chamada de uma *stored procedure* que defina uma *query* equivalente, por meio de parâmetros de execução.

A descoberta de quais seriam as funcionalidades necessárias para os *frameworks* de desenvolvimento se deu por meio do cruzamento das informações dos fornecedores das tecnologias empregadas no desenvolvimento dos sistemas (no caso, Java e .NET Framework) com as características do negócio da empresa. A técnica usada para isso foi a tabulação dos requisitos técnicos aplicáveis a todas as aplicações da empresa, para identificar a aplicação de uma arquitetura de referência, conforme Tabela 4.

TABELA 4 – Relação dos requisitos técnicos para criação do processo de uso de *frameworks* de desenvolvimento.

Requisitos Técnicos	Situação			Comentários
	Atendidos	Parcial	Não atendidos	
Autenticação	X			Todas as aplicações usam algum mecanismo de autenticação.
Autorização		X		Não foi identificado um mecanismo único para controle de acesso.
Auditoria			X	Não foi encontrado um padrão para gravar trilhas de auditoria.
Diagnóstico		X		Algumas aplicações usam rotinas para gravação de *logs* de execução.
Instrumentação			X	Nenhuma aplicação gera informações para monitorar a saúde delas.
Independência em relação a fonte de dados		X		Algumas aplicações usam rotinas de persistência onde o SGBD utilizado é definido por configuração.
Confidencialidade de dados sensíveis			X	Não foi identificado um algoritmo padrão para criptografar dados sensíveis.
Validação de dígitos de controle	X			Existem rotinas para validação de módulo 11, incluindo o duplo que é usado por CPF e CNPJ.
Validação de endereço	X			O CEP e grafia de endereços são validados contra uma base com informações vindas dos Correios.
Uso de transações para acesso a dados		X		Aplicações que necessitem dados de um determinado sistema devem utilizar as rotinas deste para usarem os dados por ele mantidos.
Recebimento de dados via arquivos	X			Foi identificado o uso de rotinas de EDI para troca de arquivos.

Controle da situação de processamento		X		Existem procedimentos onde documentos são processados por diferentes equipes até que as informações estejam prontas para entrar em uma base de dados. Mas não há um componente específico para atender este requisito.
Controle de Alçadas		X		Foi identificado um componente para controlar as aprovações em processos internos, mas ele não é compatível com as versões mais atuais das plataformas de desenvolvimento.
Integração com solução de ERP			X	Não foi identificado um padrão para expor e consumir serviços do barramento ESB da solução ERP.
Composição de Serviços			X	Não existe um padrão para implantar arquitetura orientada a serviços.
Uso de navegadores para interação com sistemas	X			As interfaces de usuários foram definidas para serem construídas como aplicações web.
Data Warehouse			X	Foi encontrado outro trabalho em andamento, voltado para implantação de um processo para BI.

A partir dos resultados obtidos na tabulação dos requisitos técnicos, foram identificadas que as seguintes funcionalidades fariam parte dos *frameworks*:

- Segurança – Mecanismos de autenticação e autorização para uso dos sistemas de informação.
- Monitoramento – Medição da saúde das aplicações e diagnóstico das causas dos problemas que eventualmente acontecerem.
- Persistência – Mecanismos para acesso e atualização de informações em fontes de dados.
- Criptografia – Abstração de algoritmos usados para proteção de dados sensíveis, segundo políticas de *compliance* da empresa.
- Negócio – Rotinas para validação dos dados mais frequentemente usados pelos sistemas de informação (por exemplo, confirmação de endereço) e de apoio para codificação de regras de negócio.

- Fluxo de Trabalho – Modelos para execução de tarefas de forma assíncrona, usados nos processamentos de lotes (*batch*) ou aqueles que necessitem de aprovações em cada etapa da execução.
- Integração – Exposição das informações como serviços para consumo por clientes externos ou em aplicações de outras plataformas que não aquela onde o *framework* é empregado.
- Apresentação – Construção das interfaces para interação dos usuários finais com os sistemas de informação.

Não fez parte do escopo para estabelecimento dos *frameworks* de desenvolvimento as necessidades de sistemas de BI, pois na época em que o trabalho correspondente a este estudo de caso se iniciou, ainda não havia uma definição de quais seriam os recursos de *Datawarehouse* que a empresa adotaria.

A descrição dos objetivos de cada uma das funcionalidades do *framework* está no documento correspondente ao Modelo de Arquitetura de Referência. Para representar essas funcionalidades foi usado um diagrama de camadas, conforme Figura 8. Nele as funcionalidades são divididas por camadas de aplicação aonde os componentes que aplicam os *frameworks* serão instalados.

FIGURA 8 – Diagrama de Camadas para as funcionalidades de *framework* de desenvolvimento.

Não foi considerada a adoção de *frameworks* de mercado devido às especificidades do negócio da empresa, identificadas durante o levantamento de requisitos, que levam à cenários não comumente encontrados em outras organizações. Também foi apontada a importância de se dominar a técnica de construção de *frameworks*, para diminuir a dependência de suporte por parte de terceiros. Outra percepção é de que com um *framework* próprio, a empresa poderá responder mais rapidamente a eventuais novidades tecnológicas que venham a ocorrer no futuro.

Neste estudo de caso, a equipe designada para construção dos *frameworks* de desenvolvimento foi a de Suporte ao Desenvolvimento. Seus membros estão entre os desenvolvedores mais qualificados da empresa, considerando critérios objetivos como número de certificações e resultados alcançados anteriormente. Além disso, como essa equipe faz parte da Gerência de Infraestrutura, ela trabalha em conjunto com os responsáveis em manter os componentes de hardware onde rodam as aplicações da empresa. Essa integração facilita a compreensão dos recursos disponíveis, de forma que os *frameworks* permitam que as aplicações funcionem de forma otimizada.

Como a equipe responsável pela construção do *framework* já fazia parte do organograma (ver Figura 7), o cronograma do projeto não contemplou a tarefa da montagem da equipe. E como foi decidida a construção de *frameworks* próprios, também não se considerou a tarefa de execução de provas de conceito com *frameworks* trazidos do mercado.

4.2.2. Especificação dos Frameworks *de Desenvolvimento*

A especificação do processo buscou ser a mais simples possível. Além de ser um subconjunto de um conjunto mais amplo correspondente à metodologia de desenvolvimento de sistemas, a facilidade de compreensão minimizaria os riscos decorrentes da sua adoção em um ambiente bastante heterogêneo, conforme descrito no Cenário Inicial. Os requisitos identificados para elaboração do processo estão apresentados na Tabela 5.

TABELA 5 – Categorização de requisitos para processo de uso de *framework* de desenvolvimento, adaptado de Dias (2010).

Visões	Categorias	Requisitos que foram Identificados
Empresa	Políticas	• Metodologia de Desenvolvimento de Sistemas; • Política de Desenvolvimento Seguro.
	Procedimentos	• Gerenciamento do ciclo de Vida das aplicações; • Integração Contínua.
	Unidades Organizacionais	• Governança de TI; • Suporte ao Desenvolvimento; • Gerência de Desenvolvimento.
	Características da Empresa	Venda de serviços para apoio à tomada de decisões.
	Stakeholders	• Áreas de negócio da empresa; • Equipes de Desenvolvimento; • Outras áreas da TI, como infraestrutura e segurança.
	Fornecedores	• IBM; • Microsoft.
	Mercado	Clientes da empresa.
Engenharia	Tipo de software desenvolvido	• Captação de informações provenientes de fontes distintas; • Consultas de dados em grande volume.
	Recursos utilizados	• IBM Websphere, CICS e DB2; • Microsoft IIS e SQL Server.
	Processos	• Codificação; • Gestão de Configuração;

		• Gestão de Liberação; • Gestão de Mudança.
Informação	Padrões	*Design Patterns*.
	Produtos de Software	• Aplicações Web; • Serviços Web; • Aplicações para processamento em lote.
Tecnologia	Tecnologias Utilizadas	• Java; • .NET Framework.
Computação	Ferramentas	• Eclipse; • Microsoft Visual Studio; • Microsoft SQL Server Management Studio; • Microsoft Team Foundation Server.

A premissa básica para se construir os *frameworks* de desenvolvimento era que fossem embasados no uso de *design patterns*. A identificação dos *design patterns* necessários foi efetuada pelo cruzamento do catálogo definido pela GoF com as funcionalidades dos componentes que fariam a composição dos *frameworks* de desenvolvimento.

Não existem regras definidas para orientar a escolha dos *design patterns*. Por isso é fundamental ter os requisitos técnicos levantados e classificados para auxiliar nessa decisão. Um exemplo foi o desejo da empresa em ter aplicações que possam trabalhar com diferentes bancos de dados sem manutenção de código, sendo que atualmente ela trabalha com três SGBDs distintos (IBM DB2, Microsoft SQL Server e Oracle). Os detalhes de como realizar essa funcionalidade estão além do escopo deste trabalho, mas o *Factory Method Pattern* consegue trazer essa independência por meio de uma simples mudança de configuração do *application server*, sem demandar recodificação.

A Tabela 6 mostra quais foram os *design patterns* identificados para construção dos *frameworks* de desenvolvimento.

TABELA 6 – Identificação dos *design patterns* aplicados pelos *framework* de desenvolvimento.

Classificação	*Design Patterns*	*Framework* para Aplicação
Criação	*Abstract Factory*	Segurança, Criptografia.
	Builder	Negócio, Integração, Apresentação.
	Factory Method	Persistência
	Prototype	Negócio
	Singleton	Segurança, Negócio, Integração, Apresentação.
Estrutural	*Adapter*	Persistência, Fluxo de Dados.
	Bridge	Segurança, Persistência, Criptografia.
	Composite	Negócio, Interface
	Decorator	Segurança, Monitoramento, Negócio, Integração, Apresentação
	Façade	Segurança, Monitoramento, Persistência, Criptografia, Negócio, Fluxo de Trabalho, Integração, Apresentação.
	Flyweight	N/A
	Proxy	N/A
Comportamental	*Chain of Responsability*	Fluxo de Trabalho
	Command	Persistência
	Interpreter	N/A
	Iterator	Negocio
	Mediator	Negócio, Integração.
	Memento	Fluxo de Trabalho
	Observer	Fluxo de Trabalho
	State	N/A
	Strategy	Persistência, Criptografia.
	Template Method	Persistência
	Visitor	Persistência, Negócio

A representação dos *design patterns* aplicados na construção dos *frameworks* de desenvolvimento foi efetuada por diagramas de classe dos componentes que implementariam os *frameworks*. A Figura 9 mostra a aplicação dos *design patterns* para o *framework* de persistência. Nela, alguns dos *patterns* aplicados estão implícitos. Por exemplo, as classes que realizam a implementação para diferentes SGBDs são façades para a APIs de acesso a dados na plataforma onde o *framework* é aplicado.

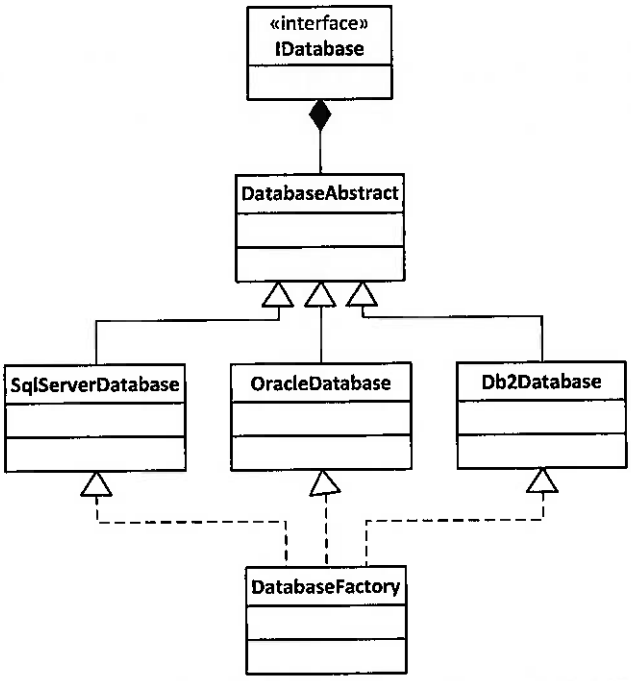

FIGURA 9 – Diagrama de classe simplificado com *design patterns* aplicados em *framework* de desenvolvimento.

Um aspecto considerado na especificação dos *frameworks* de desenvolvimento foi a constituição das equipes de desenvolvimento. Historicamente muitos analistas trabalharam de acordo com a filosofia de que para atender prazos era preciso sacrificar a qualidade. Outros tinham muita confiança em seu conhecimento e mostravam resistência em usar padrões dos quais não foram envolvidos na construção. Os *frameworks* tinham que ser ao mesmo tempo bastante fáceis de usar, para se conseguir a produtividade desejada, e completos o suficiente para inibir tentativas de desenvolvimento em paralelo de funcionalidades equivalentes às que ofereciam.

4.2.3. Construção dos Frameworks

Para apoiar o uso dos *frameworks* foram empregados alguns recursos de engenharia de software. O principal foi a construção de geradores de código, que automatizou a aplicação dos *frameworks* na construção de sistemas. Essa geração

diminuiu a execução de tarefas repetitivas na codificação de aplicações, liberando o desenvolvedor para implementar os requisitos de negócio a serem atendidos.

A introdução do processo para construção de sistemas baseado em *frameworks* de desenvolvimento foi apoiado pela adoção da ferramenta Microsoft Team Foundation Server (TFS). Esse software possui os recursos necessários para automatizar os procedimentos de gerenciamento de ciclo de vida e integração contínua, a gestão dos artefatos que compõem os *frameworks* de desenvolvimento e automação de procedimentos de QA. Um dos procedimentos de gestão de configuração que se mostrou particularmente útil foi o de versionamento, para gerência dos diferentes componentes que constituem os *frameworks* de desenvolvimento.

O desenvolvimento dos *frameworks* de desenvolvimento seguiu técnicas de TDD. Com o apoio do Sonar para gestão dos testes na plataforma Java e o TFS na gestão dos testes na plataforma .NET e *bug tracking* em ambas as plataformas, os *bugs* mais severos foram identificados e corrigidos ainda nesta etapa de construção dos *frameworks*.

4.2.4. Implantação do Processo

Como foi definido que seriam desenvolvidos *frameworks* específicos por plataforma, a priorização dos pilotos foi em função daquela na qual havia maior quantidade de sistemas previstos para desenvolvimento, que neste estudo de caso era a Java. Ademais já existiam componentes prontos para .NET Framework, faltando sistematizar o uso destes através de um processo definido.

Uma das necessidades trazidas pela implantação do processo de uso de *frameworks* de desenvolvimento foi a melhoria da gestão de liberação. Como a realização dos *frameworks* se dá através do uso de componentes que são

compartilhados entre aplicações distintas, foi necessário definir um conjunto de procedimentos para atualização desses componentes, para mitigar o risco de aplicações ficarem indisponíveis após a efetivação de mudanças.

A melhoria nos procedimentos de versionamento se mostrou essencial para a realização da gestão de liberação, pois as dependências das aplicações passaram a ser mais facilmente rastreadas e isso aumentou a segurança dos desenvolvedores, à medida que perceberam o *framework* como algo que permitia a eles focarem na programação das regras de negócio do sistema, deixando abstraídos os aspectos mais técnicos da infraestrutura onde as aplicações que codificam são executadas.

A divulgação do novo processo para uso de *frameworks* de desenvolvimento se deu por meio dos *workshops* de treinamento e pela publicação da documentação gerada. Também foi criado o procedimento para que novos integrantes das equipes de desenvolvimento fossem orientados quanto à existência do processo. Todos os envolvidos nos pilotos participaram dos *workshops* e repassaram informações úteis para ajustes em futuras versões de componentes que fazem parte dos *frameworks* de desenvolvimento.

O projeto para adoção de um processo de uso de *frameworks* de desenvolvimento foi encerrado com a implantação em produção das aplicações que fizeram parte do piloto. Dessa forma não foi realizada a tarefa de acompanhamento de resultados, mas foram estabelecidos procedimentos para cadastrar os problemas encontrados, as medidas adotadas para correção e o prazo para resolução dos problemas.

4.4. Resultados Observados

As aplicações piloto usadas para os testes integrados dos componentes que compuseram os *frameworks* de desenvolvimento apresentaram redução expressiva no número de linhas codificadas, graças ao reuso de código promovido pela aplicação desses *frameworks*. Consequentemente o tempo gasto na programação diminuiu sensivelmente.

Tomando por referência uma aplicação Java que foi refeita, as linhas de código diminuíram em mais de 50%, o que levou a uma redução de 30% no tempo gasto em codificação. Essas métricas extraídas a partir das ferramentas usadas para QA e Gestão de Projetos adotadas na empresa (Sonar e Compuware ChangePoint respectivamente) justificaram a importância de se trabalhar com padrões bem definidos.

Nas equipes que fizeram parte do piloto de adoção de *frameworks* de desenvolvimento, observou-se uma maior previsibilidade da codificação, devido ao reuso de código e melhora da qualidade do que desenvolveram em relação às entregas feitas anteriormente.

Não obstante, houve dificuldades em função da falta de experiência dos desenvolvedores em seguirem padrões. Além de certa resistência que já era previsível, constatou-se que anos de codificação sem regras definidas levou até mesmo a problemas de lógica por parte de alguns dos participantes do piloto. Nesse caso foi necessário pontuar para líderes de equipe que era necessário promover de alguma forma a reciclagem do conhecimento para aqueles que estavam com mais dificuldades em seguir o novo processo.

Outro ponto de dificuldade foi a resistência de alguns líderes, que consideravam que a participação de suas equipes no projeto poderia atrasar a entrega delas. Muitos deles não enxergaram que com um *framework* de desenvolvimento a codificação iria se acelerar e o retrabalho em função de eventuais problemas durante a execução das aplicações iria diminuir.

Testes de carga, executados segundo procedimentos estabelecidos para homologação de novas aplicações, demonstraram o bom desempenho das aplicações que fizeram parte do piloto, em questões como o uso de recursos e escalabilidade (por exemplo, transações por segundo). Esses dados confirmam que a adoção de *frameworks* de desenvolvimento deve trazer retornos em relação à qualidade dos softwares desenvolvidos na empresa. De fato, mesmo exigindo um redimensionamento dos recursos de infraestrutura, que foi necessário para as aplicações Java, elas se comportaram dentro do esperado para os requisitos de *hardware* dos servidores onde foram instaladas.

O estabelecimento de um processo definido para uso de *frameworks* endereçou as dificuldades de se estabelecer um padrão para construção de sistemas. Ao se ter um modelo documentado, utilizando os meios de comunicação à disposição da TI da empresa e tendo uma área de Governança para apoiar o processo, as resistências que aconteceram no passado foram enfraquecidas. Inclusive há gerentes que traçaram como meta para os desenvolvedores de suas equipes a utilização do *framework* de desenvolvimento.

5. CONSIDERAÇÕES FINAIS

5.1. Conclusões

O resultado do experimento descrito neste trabalho ainda não é conclusivo, pois as aplicações construídas sob o processo de uso de *frameworks* de desenvolvimento entraram recentemente em produção, portanto não há números que comprovem as melhorias esperadas. Mas os dados obtidos até o momento estão mostrando a eficácia de usar um modelo que promove o reuso de código de forma sistemática.

O uso de *design patterns* não é trivial, possivelmente por demandar um grau de abstração além do que a maioria dos desenvolvedores consegue atingir. Ao encapsulá-los em *frameworks* de desenvolvimento, resolve-se a questão de como usá-los. Mas somente com o estabelecimento de um processo de uso desses *frameworks* é possível estender os benefícios esperados do uso de *design patterns* para todos os times de desenvolvimento de uma organização.

Como a aplicação do processo se deu em equipes que trabalham com plataformas distintas, pode-se considerar que o modelo aqui proposto seria flexível o suficiente para adoção por outras organizações que enfrentem desafios similares na melhoria do desenvolvimento de sistemas de informação.

O modelo de instanciação de processos mostrou-se adequado para atender o objetivo deste trabalho. Todavia, ele precisaria ser mais difundido, para que se crie uma comunidade de especialistas no assunto que possam trocar experiências e eventualmente criar ferramentas de automação que simplifiquem a aplicação desse modelo, incluindo recursos para validação desenhos de processo.

A separação da equipe responsável pela construção e manutenção do *frameworks* e as equipes de desenvolvimento que aplicarão os mesmos demonstrou sua validade, pois eventuais dificuldades e resistências no seguimento do processo foram identificadas e controladas.

Um ponto a destacar é que o processo descrito neste trabalho não foi percebido como algo que tornasse mais lenta a construção de sistemas de informação; muito pelo contrário, ele demonstrou ser possível reduzir os tempos de entrega sem abrir mão da qualidade do código.

Neste trabalho específico, o autor contou com o apoio de uma equipe de governança de TI que baseia o seu trabalho na aplicação das melhores práticas em modelos de processo adotados pelo mercado, tais como CMMI e ITIL. A ajuda dela foi valiosa para que ocorresse a disseminação do processo.

Um desafio a vencer é o fato de que há poucos profissionais que demonstram conhecimentos tanto em disciplinas relacionadas à definição de processos como nos detalhes técnicos relacionados a um ambiente onde são construídos sistemas de informação. Esse *gap* pode ser em decorrência de que a formação do profissional voltado para a gestão de processos é bastante distinta daquele que realiza as atividades operacionais de codificação de aplicações. No estudo de caso isso foi mitigado pela formação de uma equipe multidisciplinar para desenhar o processo, mas nem todas as organizações têm recursos disponíveis para fazer o mesmo. A existência de ferramentas automatizadas para aplicar o modelo de instanciação do processo eventualmente auxiliaria a superar esse desafio.

Durante o experimento foram constatadas algumas resistências, tanto por parte daqueles que aplicariam o processo como de lideres de desenvolvimento. O motivo

dessas resistências é basicamente comportamental, possivelmente devido ao fato de que nem todos estão abertos a mudanças, talvez por não se acharem preparados para elas. Isso permite inferir que há poucas organizações com um processo formal para utilização de padrões de desenvolvimento por causa da dificuldade em formar equipes aparelhadas para trabalhar dentro de modelos estabelecidos.

Ao longo do trabalho ficou evidente que muitos desenvolvedores necessitam de reciclagem, tanto para executar as tarefas que lhes competem, como para atuarem dentro de papéis bem estabelecidos. Como o processo demandou uma maior especialização, esses desenvolvedores passaram a ser mais cobrados quanto às aptidões esperadas para construção de sistemas, tais como lógica de programação e visão sistêmica.

O trabalho foi decorrente de uma necessidade real da empresa onde se aplicou o estudo de caso. Logo faltou um maior rigor na aplicação de um método científico para resolução do problema de estabelecer um *framework* de desenvolvimento. O detalhamento de como foi instanciado o processo e as ferramentas utilizadas buscaram compensar a menor exatidão com subsídios para que outras organizações possam reproduzir os resultados aqui obtidos. A "Lei do Bem" (nº 11.196, de 21 de novembro de 2005), que prevê incentivos fiscais a empresas que desenvolverem inovações tecnológicas, quer na concepção de produtos quer no processo de fabricação e/ou agregação de novas funcionalidades ou características ao produto ou processo, pode ajudar as organizações a se aproximarem da academia para juntas obterem resultados práticos de experimentos científicos.

Para os executivos de TI, a implantação de um processo baseado no uso de *frameworks* de desenvolvimento deve ser vendida como uma estratégia para

aumentar a qualidade da codificação sem comprometer os prazos de entrega. Adicionalmente sistemas de informação melhor desenvolvidos deixam os clientes mais satisfeitos, o que pode traduzir em um maior reconhecimento dos serviços prestados pela TI.

5.2. Trabalhos Futuros

Com base nos resultados obtidos deste trabalho, seguem algumas propostas para refinar os achados desta pesquisa:

- Construção de uma ferramenta para automatizar o desenho de processos utilizando o método de instanciação. Isso pode ser através da criação de *templates* para uso em softwares existentes como Visio e BizAgi ou pela construção de um aplicativo voltado exclusivamente para essa tarefa.
- Explorar o procedimento de versionamento, que faz parte do processo de gestão de configuração, como fundamental para controlar as mudanças em ambientes onde são executados sistemas de informação.
- Relacionar de forma precisa o processo de construção de sistemas baseado em *frameworks* de desenvolvimento com metodologia de desenvolvimento orientada a objetos.
- Sistematizar o uso de *design patterns* a partir de técnicas do tipo "receita de bolo" (*cookbook*), como ferramenta para escolha daqueles mais adequados à cenários de negócio suportados por sistemas de informação. O uso de técnicas como DSL (*Domain Specific Language*) pode ser um caminho para seguir.

- Desenvolver um trabalho de acompanhamento de sistemas em produção, para comparar o comportamento daqueles construídos de forma *ad-hoc* com os que atenderam processos definidos em sua construção.
- Elaboração de um trabalho para identificar as características de membros de equipes de desenvolvimento. A proposta para esse estudo comportamental, que foge ao que foi explorado neste trabalho, se dá pelo fato de que os critérios para montagem da equipe responsável pelo *framework* de desenvolvimento são empíricos, baseados no reconhecimento profissional e não em aspectos objetivos, como alguma certificação que comprove o domínio do conceito de *design patterns*.

BIBLIOGRAFIA

TIOBE Programming Community Index for January 2012. (Janeiro de 2012). Acesso em 20 de Janeiro de 2012, disponível em TIOBE Software: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

ALEXANDER, C. (1979). *The Timeless Way of Building.* Oxford University Press.

APPLETON, B. (20 de 11 de 1997). *Patterns and Software: Essential Concepts and Terminology.* Acesso em 12 de 09 de 2011, disponível em http://www.enteract.com/~bradapp/docs/patterns-intro.html

AZIZ, A. (2003-2011). *ELMAH.* Acesso em 27 de Dezembro de 2011, disponível em Raboof.com: http://www.raboof.com/projects/elmah/

BOEHM, B., & BASILI, V. R. (January de 2001). Software Defect Reduction Top 10 List. *Computer*, 136.

BORSOI, B. (2008). *Arquitetura de Processo Aplicada na Integração de Fábricas de Software.* Tese (Doutourado em Engenharia Elétrica), Universidade de São Paulo.

BRINKKEMPER, S. (1996). Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology*, 275-280.

CHANG, C.-H., LU, C.-W., & HSIUNG, P.-W. (2011). Pattern-Based Framework for Modularized Software Development. *Information and Software Techonology*(53), 312.

COPLIEN, J. O. (s/d). *Design Pattern Definition.* Acesso em 26 de Dezenbro de 2011, disponível em The Hillside Group: http://hillside.net/component/content/article/50-patterns/222-design-pattern-definition

CWALYNA, K., & ABRAMS, B. (2009). *Framework Design Guidelines* (2a. ed.). Addison-Wesley.

DIAS, L. D. (2010). Método de Instanciação de uma Arquitetura de Processos Aplicado em Fábrica de Software. *Dissertação (mestrado em Engenharia de Computação) Orientador: Jorge Luis Risco Becerra.* Universidade de São Paulo.

ESPINDOLA, R. S., & MAJDENBAUM, A. e. (2004). Uma Análise Crítica dos Desafios para Engenharia de Requisitos em Manutenção de Software. *Programa de Pós-Graduação em Ciência da Computação PUC-RS*, p. 2.

ESPOSITO, D., & SALTARELLO, A. (2009). *Microsoft .NET: Architecting Applications for the Enterprise.* Microsoft Press.

FILHO, I. m., OLIVEIRA, T. C., & LUCENA, C. J. (2004). A framework instantiation approach based on the Features Model. *The Journal of Systems and Software*, 333-349.

GAMMA, E., HELM, R., JOHNSON, R., & VLISSIDES, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley.

KHOMH, F., & GUÉHÉNEUC, Y.-G. (2008). Do Design Patterns Impact Software Quality Positively? IEEE.

MESO, P. e. (Verão de 2006). Agile Software Development: Adaptative Systems Principles and Best Practices. *Information Systems Management*, p. 19.

MILLETT, S. (2010). *ASP.NET Design Patterns.* Wrox.

MU, H., & JIANG, S. (2011). Design Patterns in Software Development. IEEE.

PANIGASSI, R. (2007). Método para definição e modelagem de processos de fábrica de software usando RM-ODP e BPM. *Dissertação (mestrado em Engenharia Elétrica). Orientador: Jorge Luis Risco Becerra*. Universidade de São Paulo.

SHERIF, K., & VINZE, A. (2003). Barriers to adoption of software reuse A qualitative study. *Information & Management*(41), 159-175.

WOO, F., MIKUSAUKAS, R., BARLETT, D., & LAW, R. (2006). Is OO the Systems Development Technology for Your Organization? *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*. IEEE Computer Society.