METODOLOGIA DE IMPLEMENTAÇÃO DE SISTEMAS ORIENTADOS A OBJETOS UTILIZANDO FRAMEWOKS

Cleverson Sacramento de Oliveira *

Resumo: *Ao produzir sistemas de computadores, são enfrentados muitos problemas que interferem direta e indiretamente na produtividade do desenvolvimento de softwares. Muitas dessas dificuldades são decorrentes da falta de padronização e de erros gerados na etapa de implementação. Para solucioná-las, aposta-se na criação de uma metodologia de implementação de sistemas e na utilização de um framework – combinação de componentes que auxiliam no desenvolvimento de aplicações – com o objetivo de aumentar a produtividade. A metodologia e o framework devem ser projetados, almejando a facilidade de mudanças, possibilitando a contínua evolução de ambos. Para desenvolver a metodologia e o framework, foi necessária a definição de um projeto-piloto com uma baixa complexidade inicial. A implementação do framework se deu com a utilização de algumas técnicas já consagradas para garantir a qualidade final do projeto. Pôde-se notar, com a análise dos dados coletados, que os elementos padronizados e encapsulados na estrutura do framework contribuíram para a redução significativa do tempo despendido na elaboração do projeto-piloto, bem como o aumento da produtividade da equipe. Portanto esta solução mostrou-se viável para o domínio de problema estudado.*

Palavras-chave: Engenharia de Software; Metodologia de implementação de sistemas; Framework

1 INTRODUÇÃO

A busca do aumento da produtividade na confecção de sistemas é um grande desafio para a Engenharia de Software que, para Sommerville (2003, p.5), "[...] é uma disciplina da engenharia que se ocupa de todos os aspectos da produção de *software*, desde os estados iniciais [...] até a manutenção [...]". É fácil observar que as empresas do ramo estão a cada dia investindo mais em melhorias no processo e qualidade do produto, utilizando padrões e modelos bem difundidos no mercado, tais como os existentes no *International Organization for Standardization* (ISO) e o *Capability Maturity Model* (CMM). Porém somente estes conceitos não são suficientes para atingir tal meta, deve-se ressaltar que práticas próprias e experiências adquiridas são fatores também relevantes neste processo. (FERNANDES; TEIXEIRA, 2004, p.27)

As práticas próprias de uma empresa de desenvolvimento de *software* devem ser bem documentadas e definidas, viabilizando um rápido aprendizado e facilitando a sua aplicação cotidiana. A Engenharia de Software preocupou-se em definir a utilização de modelos para o desenvolvimento de sistemas, dando origem à Metodologia de Desenvolvimento de Sistemas (LEMME FILHO, 2003). Esta por sua vez perpassa por várias etapas, dentre elas, a que é abordada neste trabalho: a etapa de implementação, onde todo o projeto é codificado em linguagem de programação.

A fim de elevar ainda mais a produtividade na implementação, é fundamental agregar as vantagens oferecidas pelos *frameworks* (MARKIEWICZ; LUCENA, 2001) com uma metodologia de implementação de sistemas. Este será o foco desta pesquisa: analisar problemas específicos da etapa de codificação, buscando soluções práticas obtidas a partir da definição de

* Acadêmico do Curso de Bacharelado em Informática da Universidade Católica do Salvador – UCSal. E-mail: zyc@ig.com.br. Orientador: Professor M.Sc. Josemar Rodrigues de Souza. E-mail: josemar@ucsal.br.

metodologias aliadas a *frameworks,* em busca do aumento da produtividade. Para tanto, alguns aspectos de projeto imprescindíveis (PRESSMAN, 1995, p.420) serão abordados no decorrer do texto, buscando soluções viáveis para os problemas acarretados pela falta de padrões na etapa de implementação de sistemas. O sistema de métricas utilizado será o *Function Point Analysis* (FPA) (IFPUG, 2004).

Antes mesmo de tratar da metodologia de implementação de sistemas, é fundamental ter claro o que vem a ser metodologia. Segundo Leme Filho (2003, p.5), o termo metodologia "[...] refere-se a uma série de procedimentos, definidos previamente [...] e que precisam ser executados em uma determinada seqüência, para atingir o resultado final esperado", assim, pode-se evitar o aspecto rudimentar nesta etapa de desenvolvimento. Dispõe-se de alguns meta-modelos que servem como auxílio para codificação de diferentes contextos, e o enfoque deste material se dará no modelo Orientado a Objetos (OO) por motivos relacionados a características inerentes a este paradigma (PETERS; PEDRYCZ, 2001, p.226).

Pertencente ao processo de desenvolvimento de sistemas, a etapa de implementação transforma toda a coleta de dados e modelos desenvolvidos em um programa "palpável", algo funcionalmente prático. É neste momento que o processo de codificação acontece, convertendo idéias abstratas em linguagem artificial (PRESSMAN, 1995, p.47). A Figura 1 mostra a etapa de implementação (codificação) em destaque.

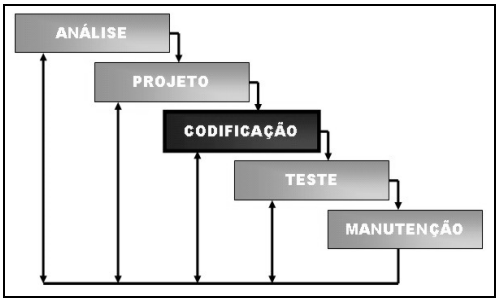

Figura 1 – O ciclo de vida clássico (PRESSMAN, 1995, p.33) adaptado

Visando abstrair níveis de complexidade mais elevados para resolução de problemas na etapa de implementação, deve-se, de alguma maneira, encapsular a codificação em componentes. Com isso, torna-se fácil "[...] liberar o programador de pensar em um nível inadequado de detalhes" (PETERS; PEDRYCZ, 2001, p.8-9), para que ele possa focar toda a atenção na lógica do negócio, buscando atingir uma melhor solução. Para tanto, este trabalho se utilizar das vantagens oferecidas por um *framework* que, traduzindo, significa "estrutura" (ALTA VISTA, 2004). Entretanto, buscando ser mais específico no tema em questão, pode-se defini-lo como sendo "[...] uma combinação de componentes [...] que simplifica a construção de aplicações e que pode [e deve] ser conectado a uma aplicação." (PETERS; PEDRYCZ, 2001, p.3).

Ao projetar *frameworks*, é preciso ficar claro que eles "[...] são construídos para flexibilidade e generalidade, tentando abranger um domínio inteiro em vez dos problemas particulares" (MARKIEWICZ; LUCENA, 2001, tradução nossa). Portanto é necessário fazer um levantamento das freqüentes dificuldades enfrentadas na implementação das aplicações, evitando o desperdício de recursos alocados em tarefas que serão pouco reutilizadas.

Ao desenvolver um projeto de *software*, é comum deparar-se com situações problemáticas já vivenciadas e já solucionadas. Costuma utilizar-se destas soluções bem sucedidas para resolver problemas futuros, apenas readaptando-as ao novo contexto. Lemme Filho afirma que "construir ou codificar um aplicativo segundo uma boa metodologia é poder aplicar técnicas já consagradas para garantir a qualidade do trabalho e do produto final" (2003, p.66). Facilmente nota-se a veracidade desta assertiva na fase de implementação do sistema,

ganhando-se tempo, ao evitar a recriação de uma solução já conhecida e confiança, por valer-se de uma experiência prévia (ALUR; CRUPI; MALKS, 2002, p.3). Portanto foram utilizados padrões de projeto (GAMMA et al, 2000) para a elaboração da solução proposta.

Podem-se facilmente detectar alguns problemas que oneram o processo de implementação de sistemas e que, conseqüentemente, impactam no prazo de finalização e entrega do produto. Segundo Chrissis, Konrad e Shrum (2003, p.3), estes problemas têm tendência a acentuar-se cada vez mais devido ao aumento da complexidade dos sistemas, diminuição dos prazos e a redução nos custos de um projeto. Enfim, devem-se almejar produtos mais competitivos. Os erros e a falta de padronização no código-fonte, na etapa de implementação, são também grandes causadores deste transtorno, concebendo projetos com aspectos manufaturados que prejudicam a manutenibilidade do código.

Por mais bem especificada que uma metodologia pareça ser, ocorrerão modificações no decorrer da sua implantação devido a novas necessidades não contempladas, padrões pouco aplicáveis à prática e ajustes na sua especificação. O mesmo acontecerá com o projeto de *framework*. Daí surge a necessidade de considerar mais atentamente uma relação entre eles: a flexibilidade para modificação. É muito importante que os incrementos sejam "[...] pequenos e cuidadosamente selecionados, visando os incrementos futuros" (PETERS; PEDRYCZ, 2001, p.14), podendo assim manter o controle sobre o impacto da mudança. Esta integração entre a metodologia e o *framework*, doravante denominada metodologia/*framework*, será discutida em capítulos subseqüentes.

2 AMBIENTE DE DESENVOLVIMENTO

Para desenvolver a solução, é preciso focar-se no domínio do problema: desenvolver uma solução que colabore para a padronização e facilitação do processo de implementação de aplicações WEB para uso comercial, priorizando a praticidade de implantação da aplicação em provedores de hospedagem gratuita. Esta solução deverá ser facilmente escalável, reutilizável e com baixo custo de aprendizado, viabilizando o aumento de produtividade. É indispensável utilizar o paradigma OO no desenvolvimento do *framework*, cabendo a cada aplicação que faça uso deste seguir uma metodologia de implementação de sistemas.

O *framework*-piloto começou a ser projetado para a plataforma *Java 2 Enterprise Edition* (J2EE), versão 1.3 (SUN, 2004), devido a sua grande aceitação no mercado para aplicações de grande complexidade técnica (PETERS; PEDRYCZ, 2001, p.484) nas intranets. Porém, ao tentar implantar o projeto-piloto na Internet, foram encontrados diversos percalços que inviabilizaram a continuidade evolutiva da solução, para os fins desejados no momento (jan. 2003) tais como: a precariedade no serviço de hospedagem e implantação das aplicações J2EE nos poucos sites *hostfree* (servidor de hospedagem gratuita) que suportam a tecnologia; poucos profissionais capacitados no mercado; público-alvo (empresas de desenvolvimento de pequeno e médio porte) não preparado ou indisposto para investir na tecnologia.

A plataforma *Active Server Pages* (ASP), versão 3.0 (MSDN, 2004a), com bastante aceitação no mercado, agregou uma gama de fatores interessantes que suprem os requisitos supracitados. Normalmente, quando se pensa em aplicações desenvolvidas nesta tecnologia, logo se associam códigos de programa misturados com linguagem de marcação (HTML), ou pensa-se em soluções mal organizadas de difícil manutenibilidade. Porém, usufruindo-se de padrões de projeto e técnicas de orientação a objetos, pode-se explorar o recurso de utilização de classes para extrair e beneficiar-se com as vantagens de se trabalhar com este paradigma.

Uma versão resumida da linguagem de programação *Visual Basic*, o *VBScript* (MSDN, 2004b) incorpora o recurso de criação de classes, dando suporte à manipulação de objetos. Apesar de não apresentar um bom controle de erros e exceções, esta linguagem foi escolhida

neste trabalho para ser aplicada no *server-side*, graças a sua integração com a técnica de orientação a objetos. Não cabe a este projeto de pesquisa discutir esta questão polêmica e duradoura: o *Visual Basic* é ou não é uma linguagem OO? Alguns autores, como Cooper (2001, p.57), defendem a idéia de que o *Visual Basic* seja realmente uma linguagem orientada a objetos. Porém esta linguagem não apresenta algumas características como herança e polimorfismo, por exemplo, descaracterizando-a como uma linguagem OO. Enfim, é fato que os recursos ofertados pela linguagem que dão suporte à manipulação de objetos serão exaustivamente explorados e utilizados para implementar modelos e padrões para soluções orientadas a objetos. Neste trabalho, o recurso de herança de classes será substituído pela delegação, conforme proposto por Gamma et al (2000, p.36-37).

Para desenvolver a metodologia/*framework*, é necessário eleger um projeto-piloto. Para tanto, esta aplicação não deve possuir um nível de complexidade (PETERS; PEDRYCZ, 2001, p.484) inicial muito elevado, nem deve estar sujeita a prazos curtos de finalização. Para uma empresa, é ideal que este seja um projeto interno. Tudo isto é necessário para que se possa trabalhar sem preocupações de reestruturar a aplicação em função de uma mudança muito significativa na estrutura da medologia/*framework*, que comprometeria o prazo de finalização do aplicativo. Sendo um projeto interno, tem-se a vantagem de modificá-lo no decorrer de seu desenvolvimento sem uma prévia aceitação do cliente, agregando funcionalidades que servirão como casos de teste para implantação de novas abstrações no *framework* ou reformulação da metodologia. Em outras palavras, pode-se calibrar a complexidade da aplicação, buscando novos problemas para serem reunidos e solucionados pela metodologia/*framework*.

À primeira instância, teve-se a idéia de implementar um sistema de gerenciamento de conteúdo para portais, adotando-o como projeto-piloto. Por motivos de pressões sobre o prazo e solicitações de novas implementações para o portal, este deixou de ser o projeto-piloto e seguiu seu ciclo de vida com uma versão estagnada da metodologia/*framework*. Logo em seguida, surgiu uma proposta para elaboração de um sistema de fichamento, sugerido pelo professor Luiz Morais da disciplina Introdução à Metodologia de Pesquisa (MPI) desta instituição. A definição deste sistema é baseada em conceitos da Metodologia de Pesquisa Científica, definidas por autores consagrados neste assunto, que tem como objetivo, segundo os próprios Lakatos e Marconi (1992, p.51), atender aos seguintes itens: "[...] (a) identificar as obras; (b) conhecer seu conteúdo; (c) fazer citações; (d) analisar o material; (e) elaborar críticas".

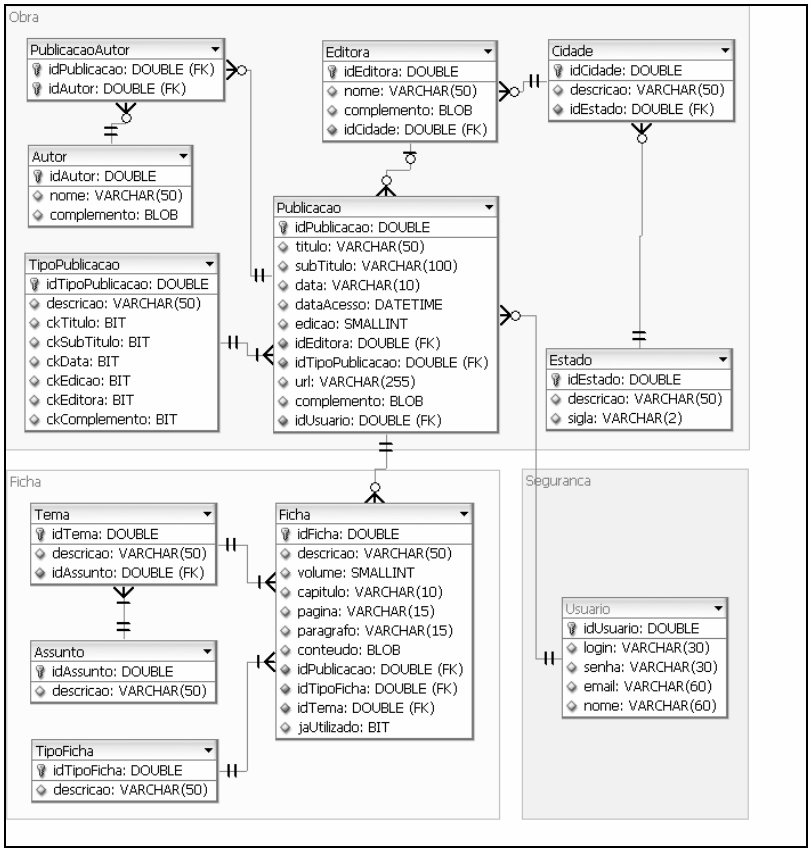

Figura 2 – Diagrama de Entidades e Relacionamentos do projeto-piloto

Para uma melhor compreensão deste sistema, a Figura 2 apresenta o Diagrama de Entidades e Relacionamentos gerado a partir do *DBDesigner* (DBDESIGNER, 2004), representando a modelagem do banco de dados utilizado pelo projeto-piloto. O modelo foi implementado no *Microsoft Access* devido a sua praticidade de implantação. O usuário poderá cadastrar estados, cidades, editoras, autores, assuntos, temas e publicações (livros, revistas, artigos, sites, etc.) possibilitando o fichamento. O sistema deve permitir acesso multi-usuário, controle de segurança, telas de cadastro, busca de fichas e obras. Deve prover também telas de consulta refinada, facilitando ao máximo para o usuário seguir as regras e padrões de formatação de referências bibliográficas e citações.

3 METODOLOGIA E FRAMEWORK PROPOSTOS

Para resolver dois problemas clássicos enfrentados (falta de padronização e erros na etapa de implementação) que desencadeiam uma série de outros problemas, será adotada a implantação de metodologia e *framework*. As dificuldades desencadeadas são, por exemplo: dependência do código com o programador que o criou, dificuldade de manutenção, aumento no custo do produto, não-cumprimento do prazo, re-trabalho e disponibilidade de diversas soluções para resolver o mesmo problema. A metodologia tratará as dificuldades geradas pela falta de padronização, enquanto o *framework* buscará diminuir os erros na etapa de implementação, agregando muitas facilidades a esta fase de desenvolvimento do *software*.

Alguns procedimentos foram definidos para alavancar o desenvolvimento da metodologia de implementação de sistemas proposta, dentre eles está: (i) definir padrões de nomenclatura de variáveis, constantes, classes, atributos, métodos e arquivos; (ii) organizar a hierarquia de pastas

e pacotes; (iii) elaborar um método de documentação do código-fonte; (iv) desenvolver o projeto fundamentado nas facilidades e estrutura do *framework*. Deve-se observar que os padrões são baseados na estrutura do *framework*, ficando claro o acoplamento entre eles (metodologia/*framework*).

É fundamental, para definição de aspectos na etapa de implementação, já dispor de decisões referentes à fase de projeto. O foco dado a estas decisões será para as características pertencentes à etapa de implementação, conforme o escopo deste trabalho.

Seguindo um padrão de nomenclatura de itens concernentes à etapa de implementação do sistema, torna-se mais compreensível o código-fonte a ponto de facilmente conseguir diferenciar uma variável de uma constante, por exemplo. Deve-se utilizar a padronização para subentender a funcionalidade de um elemento a partir de seu nome, agregando informações sobre sua funcionalidade nas "entrelinhas". Quanto mais bem especificadas são as regras, menos flexível e subjetiva se tornará a aplicação.

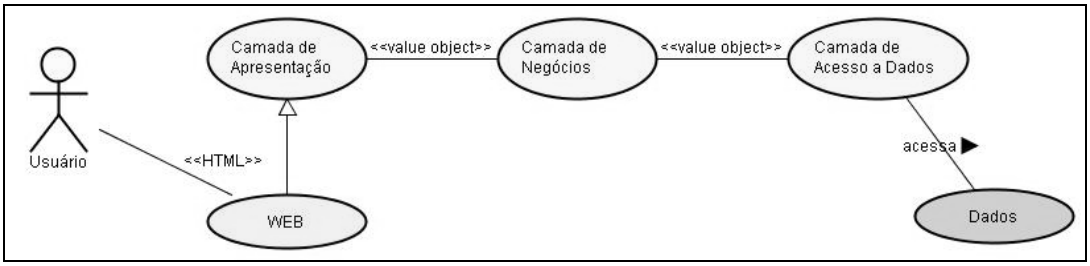

Figura 3 – Visão geral do modelo multicamadas

A fase de implementação será baseada em um modelo de camadas, almejando a separação entre a apresentação (*front-end*) e a lógica de negócios (*back-end*). Sendo assim, poder-se-á elevar o nível de modularização e manutenibilidade (PRESSMAN, 1995, p.70) do sistema, já que a "[...] lógica da aplicação é dividida em componentes de acordo com sua função [...]" (BODOFF et al, 2002, p.2). Entenda-se modularização de *software* como sendo a divisão lógica em componentes que executam funções específicas (PRESSMAN, 1995, p.419). A divisão lógica e física será feita em basicamente três sedimentos, conforme a Figura 3: apresentação, negócios e acesso a dados.

Assim como em uma biblioteca, é importante manter a organização, ao armazenar as obras ou estabelecer regras para guardar fichas, buscando uma maior eficiência no resgate do material. Devem-se separar classes por características comuns, agrupando-as logicamente em pacotes e fisicamente em pastas ou arquivos, como o *Java Archive* (JAR) (SUN, 2004), por exemplo. Para manter uma organização coerente, deve-se seguir a mesma estrutura tanto para a distinção lógica quanto física, ou seja, a estrutura dos pacotes deve ser similar à estrutura das pastas.

Para estruturar a lógica em códigos de programação, o desenvolvedor baseia-se em teorias estudadas e em experiências próprias. Mesmo com toda a padronização estabelecida pela metodologia, há sempre espaço para agregar características individuais do programador. Visando amenizar tal problema, deve-se elaborar uma documentação a partir do comentário do código-fonte, a partir de uma descrição das classes, atributos e métodos, seguindo o padrão especificado por uma ferramenta de geração automática da documentação (VBDOX, 2002).

Para garantir o acúmulo de experiências já vivenciadas em dada empresa, é preciso encapsulá-las no *framework*. Estas experiências são derivadas da cultura da empresa e envolvem muitas variáveis (ALUR; CRUPI; MALKS, 2002, p.3). Assim, o *framework* acumulará características próprias se tornando uma ferramenta ímpar. Ao implementar estas experiências, é

importante fazer uso de padrões de projeto, que serão considerados como parte integrante do *framework*, como mostra a Figura 4.

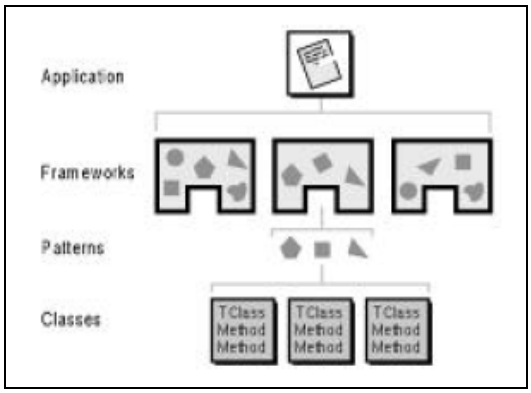

Figura 4 – Partes integrantes de um *framework* (IBM, 2000, p.13)

Evitando que o programador se detenha em minúcias da arquitetura em utilização, deve-se desenvolver uma abstração para facilitar, agilizar e padronizar a codificação. Neste caso, a abstração será criada para a arquitetura WEB, seguindo o padrão *View Helper* (ALUR; CRUPI; MALKS, 2002, p.136). Este procedimento permite que o código *server-side* seja separado do *client-side*, abandonando o aspecto rudimentar que geralmente se associa a programas desenvolvidos em ASP.

A implementação de segurança é bastante difundida no desenvolvimento de aplicações comerciais e, para evitar a recodificação a cada novo sistema, esta deve ser incorporada ao *framework*. Buscando uma centralização no tratamento de autenticação de usuários e permissão de acesso, recomenda-se utilizar o padrão de projeto *Front Controller* (ALUR; CRUPI; MALKS, 2002, p.165), originado do *Smalltalk* (linguagem de programação OO) e atualmente explorado no *Jakarta Struts* (GOODWILL, 2002, p.3).

4 RESULTADOS OBTIDOS

Durante o desenvolvimento do projeto, puderam-se observar alguns ganhos significativos na etapa de implementação. Com o auxílio do padrão e da ferramenta de documentação técnica do código-fonte, teve-se, ao final do processo, uma referência bem organizada à codificação do projeto-piloto e do *framework*. Assim, resolveu-se o problema de implementações mal documentadas e, seguindo a metodologia, obteve-se uma estrutura de armazenamento de classes bem definida, separando-as de acordo com a sua funcionalidade, de maneira modular (PRESSMAN, 1995, p.427).

Ao término do processo de implementação do projeto-piloto e do *framework*-piloto, foi feito um teste de atualização no projeto-piloto abortado: o sistema de gerenciamento de conteúdo. Este projeto possuía uma versão mais antiga da metodologia e do *framework* que a apresentada neste trabalho, porém sua implementação foi projetada com base nas mesmas premissas. Daí então surgiu a curiosidade de submetê-lo a um processo de atualização do *framework* sem modificação na metodologia, com a finalidade de testar o grau de acoplamento entre eles. O processo despendeu quatro horas para migração da nova versão do *framework*.

O projeto-piloto foi inicialmente desenvolvido por uma equipe composta por quatro programadores, dentre eles, dois experientes em ASP e dois que nunca haviam desenvolvido nesta arquitetura. O processo de treinamento formal sobre a metodologia/*framework* durou cerca de uma hora e aproximadamente mais duas horas de acompanhamento prático individual

(treinamento informal). Os treinamentos foram ministrados pelo programador experiente que dominava a metodologia/*framework*. A etapa de implementação se deu sem maiores intercorrências, sendo possível facilmente estabelecer a divisão de tarefas entre os desenvolvedores, que tranqüilamente se adaptaram, graças à metodologia.

No projeto-piloto, foi de suma importância medir o ganho obtido com a utilização dos conceitos apresentados neste trabalho. A contagem foi baseada no sistema de métrica FPA, aplicada ao projeto-piloto, no seu processo inicial de desenvolvimento, onde havia a maior carga de trabalho. Neste momento já se dispunha de uma versão estável da metodologia do framework e do projeto-piloto. Algumas tabelas da base de dados, como a de usuário e de estado, não eram alimentadas pelo sistema, portanto foram consideradas como arquivos de interface externa no cálculo do FPA.

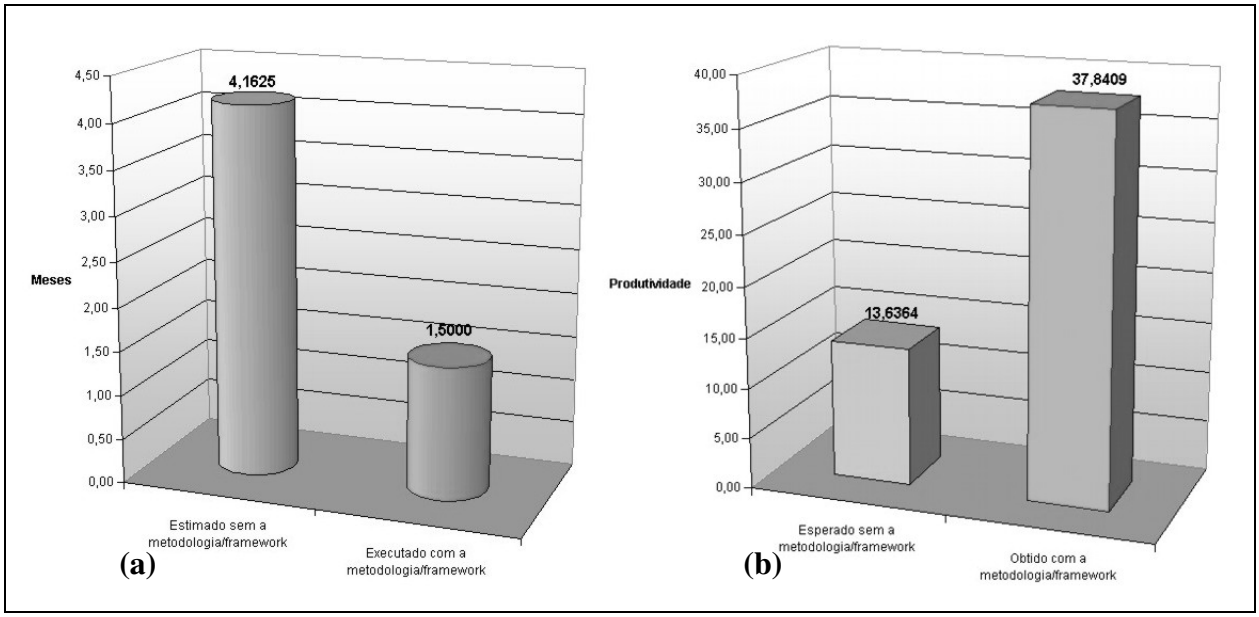

Figura 5 – Redução de tempo e aumento de produtividade

O sistema foi mensurado em 228,48 *Function Point* (FP), com prazo de conclusão estimado em quatro meses e três dias aproximadamente, dispondo de quatro programadores com dedicação de três horas diárias, em vinte dias por mês. Estima-se que cada pessoa produza, em média, uma unidade de FP em quatro horas e vinte e quatro minutos no desenvolvimento de aplicações WEB (BFPUG, 2004). Entretanto o tempo real para a conclusão do projeto-piloto foi de uma semana para a análise do sistema (levantamento de dados e modelagem), um mês de implementação e uma semana de testes. Portanto o tempo total gasto para desenvolver o sistema foi de um mês e duas semanas, que representa 36,036% do tempo total estimado, utilizando a metodologia/*framework* sugerida neste trabalho. O gráfico ilustrado na Figura 5a exibe o tempo despendido para a conclusão do projeto-piloto, comparando o valor estimado e o efetivamente gasto.

O gráfico comparativo da Figura 5b exibe o ganho de 177,5% de produtividade, de acordo com a fórmula proposta por Pressman (1995, p.66), gerando uma unidade de FP em uma hora e trinta minutos de trabalho por pessoa.

5 CONCLUSÃO

Com o uso de padrões e com a redução de erros na etapa de codificação, tornou-se evidente o aumento significativo da produtividade obtido com utilização da metodologia/*framework* proposta, para este domínio de problema. O processo de migração da metodologia e do *framework*, no projeto-piloto abortado, mostrou-se viável, gerando pouco impacto à regra de negócio da aplicação.

Ao adotar uma metodologia/*framework*, as implementações cotidianas são abstraídas para o desenvolvedor, diminuindo assim o risco de erros na fase de implementação. Baseando-se em padrões para implementar soluções, o aspecto manufator é deixado de lado, porém não deve ser aplicado com exacerbo. É importante também conservar espaço para a criatividade do programador e possibilitar contornos de futuros obstáculos não contemplados pelo *framework*, a partir de novas implementações que servirão como sugestões para melhoria contínua da ferramenta.

Como sugestão para trabalhos futuros, pode-se implementar algumas soluções que visem o aumento da produtividade, descritas a seguir. Buscando facilitar a interface para o desenvolvimento de aplicações, pode-se construir um ambiente de desenvolvimento com geração de códigos de forma automática, com interfaces plugáveis. Explorando os recursos de segurança, convém agregar criptografia às informações que podem ser utilizadas no momento da persistência dos dados, de forma automática. Para aproveitar ainda mais as vantagens oferecidas pela metodologia proposta, pode-se implementar o *framework* em outras linguagens de programação.

6 REFERÊNCIAS

ALTA VISTA, **Tradução Babel Fish**. Disponível em: <http://babelfish.altavista.com/babelfish/tr/>. Acesso em: 31 de mar. 2004.

ALUR, Deepak; CRUPI, John; MALKS, Dan. **Core J2EE Patterns**: best practices and design strategies. 2. ed. Upper Saddle River: Prentice Hall. 2002.

BFPUG, **Brazilian Function Point Users Group**: qual a produtividade do java? Rio de Janeiro. Disponível em: <http://www.bfpug.com.br/Produtividade_Java.htm>. Acesso em: 07 de jun. 2004.

BODOFF, Stephanie et al. **Tutorial do J2EE**: a série java enterprise edition autorizado. Rio de Janeiro: Campus. 2002.

CHRISSIS, Mary Beth; KONRAD, Mike; SHRUM, Sandy. **CMMI**: guidelines for process integration and product improvement. Boston: Addison Wesley. 2003.

COOPER, James W.. **Design Patterns and Object Oriented Programming**: in visual basic 6 and VB.NET. [S.I.]: IBM Thomas J Watson Research Center. 2001.

DBDESIGNER. Version 4.0.5.4. [S.I.]: FabForce. Disponível em: <http://www.fabforce.net/downloads.php>. Acesso em: 10 de jun. 2004.

FERNANDES, Aguinaldo Aragon; TEIXEIRA, Descartes de Souza. **Fábrica de Software**: implantação e gestão de operações. São Paulo: Atlas. 2004.

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

GOODWILL, James. **Mastering Jakarta Struts**. Indianapolis: Wiley Publishing. 2002.

IBM. **Building Object-Oriented Frameworks**. [S.I.]: IBM. 2000.

IFPUG, **International Function Point Users Group**. Princeton. Disponível em: <http://www.ifpug.org>. Acesso em: 19 de mai. 2004.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Metodologia do Trabalho Científico**. 4. ed. São Paulo: Atlas. 1992.

LEME FILHO, Trajano. **Metodologia de Desenvolvimento de Sistemas**. Rio de Janeiro: Axcel Books. 2003.

MARKIEWICZ, Marcus Eduardo; LUCENA, Carlos J.P. **Object Oriented Framework Development**. [S.I.]. ACM Crossroads, 2001. Disponível em: <http://www.acm.org/crossroads/xrds7-4/frameworks.html>. Acesso em: 23 fev. 2004.

MSDN. **ASP**: build-in objects. [Seatle]: Microsoft Coporation. Disponível em: <http://msdn.microsoft.com/library/en-us/iissdk/iis/ref_vbom_.asp>. Acesso em: 16 de mar. 2004.

_____. **VBScript Reference**. [Seatle]: Microsoft Coporation. Disponível em: <http://msdn.microsoft.com/library/en-us/script56/html/vtoriVBScript.asp>. Acesso em: 10 de mar. 2004.

PETERS, James F.; PEDRYCZ, Witold. **Engenharia de Software**: teoria e prática. Rio de Janeiro: Campus. 2001.

PRESSMAN, Roger S.. **Engenharia de Software**: uma aplicação prática. 3. ed. São Paulo: Makron Books. 1995.

SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Addison Wesley. 2003.

SUN MICROSYSTEMS. **Java**. [Santa Clara]. Disponível em: <http://java.sun.com>. Acesso em: 9 de mar. 2004.

VBDOX: Visual Basic Documentation Generator. [S.I.]: SourceForge, 2002. Disponível em: <http://vbdox.sourceforge.net>. Acesso em: 8 de mar. 2004.