UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GUILHERME SILVA DE LACERDA

FrameworkDoc: ferramenta de documentação e geração de artefatos de software

Trabalho de Conclusão apresentado como requisito parcial para a obtenção do grau de Mestre em Informática

Prof. Dr. Marcelo Soares Pimenta
Orientador

Porto Alegre, março de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lacerda, Guilherme Silva de

FrameworkDoc: ferramenta de documentação e geração de artefatos de software / Guilherme Silva de Lacerda – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

77 f.:il.

Trabalho de Conclusão (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Marcelo Soares Pimenta.

1. Engenharia de Software. 2. *Frameworks.* 3. Documentação de Software. 4. Geração de Artefatos. I. Pimenta, Marcelo Soares. II. Título.

AGRADECIMENTOS

Este trabalho não teria sido possível sem a ajuda indispensável de tantas pessoas.

À Deus, por me dar vida e saúde para cumprir minha missão;

À família, por ser o alicerce da minha realização;

Aos meus pais, que me deram a orientação e apoio para o caminho certo;

Aos meus amigos, pelo apoio e motivação recebida durante o período do curso;

À Juliana Bianchin, por iluminar o meu caminho com sua alegria e me fazer muito feliz.

Ao grande mestre Professor Marcelo Pimenta, com quem tive o prazer de conhecer, ser seu aluno, ser seu orientado e amigo.

Ao irmão, colega e sócio Claudimir Zavalik, pelos constantes comentários e importantes contribuições.

Aos colegas do curso de Mestrado, que conheci durante esta caminhada e que são meus amigos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ASP	*Active Server Pages*
BD	Banco de Dados
CSS	*Cascading Style Sheets*
DAO	*Data Access Object*
DLL	*Dynamic Link Library*
DSDM	*Dynamic Systems Development Method*
DTD	*Data Type Definition*
E-R	Entidade-Relacionamento
FTP	*File Transfer Protocol*
GOA++	*Gerente de Objetos Armazenados*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
I18N	*Internacionalization*
IDE	*Integrated Development Environment*
J2EE	*Java 2 Enterprise Edition*
J2ME	*Java 2 Micro Edition*
J2SE	*Java 2 Software Edition*
JCP	*Java Community Process*
JDBC/ODBC	*Java Database Conectivity/Open Database Conectivity*
JDK	*Java Development Kit*
JNI	*Java Native Interface*
JSP	*JavaServer Pages*
MA	*Modelagem Ágil*
MVC	*Model-View-Controller*
OMT	*Object Modeling Technique*

OO	Orientação a Objetos
OOSE	*Object-Oriented Software Enginnering*
PADS	Processo APOENA de Desenvolvimento de Software
PDF	*Portable Document Format*
PHP	*PHP Hipertext Preprocessor*
PMR	Processo e Modelos Recomendados
RTF	*Rich Text Format*
RUP	*Rational Unified Process*
SGBD	Sistemas Gerenciadores de Banco de Dados
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
VO	*Value Object*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*
XP	*eXtreme Programming*
XSLT	*eXtensible Style Language Transformation*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Atualmente, um dos grandes desafios para qualquer desenvolvedor de software é projetar um sistema que reutilize ao máximo elementos de código e de projeto existentes, visando diminuir o tempo e o esforço exigidos na produção do software. Entre as inúmeras formas de possibilitar reuso no contexto do desenvolvimento segundo o paradigma da orientação a objetos, destaca-se a abordagem de *frameworks.*

A grande importância da documentação de software utilizada no processo de desenvolvimento aliada às características de *frameworks* serviram como motivação para este trabalho. A documentação dentro do processo de desenvolvimento de software não faz parte de uma fase definida, mas ocorre durante toda sua existência, em paralelo com outras fases do ciclo de vida. A abordagem de *frameworks* dentro deste contexto enfoca o tratamento de *templates* e definições das características dos artefatos de software (incluindo não somente código mas também produtos de análise, projeto, *frameworks*, componentes, diagramas, entre outros), facilitando e acelerando o processo de documentação.

Um *framework*, devido a suas características peculiares que serão examinadas e explicitadas no trabalho, contém uma série de informações que podem, além de apoiar a documentação, ser úteis para produção de outros artefatos (por exemplo, planejamentos de teste, *scripts* de bancos de dados, padrões de codificação, entre outros) do processo de desenvolvimento. Assim, em um processo de desenvolvimento evolutivo, que utiliza a geração de artefatos como recurso, a manutenção pode ser integralmente realizada somente na especificação e não diluída nos artefatos gerados.

O objetivo deste trabalho é investigar, propor e desenvolver uma ferramenta de documentação e geração de artefatos de software, denominado *FrameworkDoc*. O termo documentação de software aqui utilizado se refere a documentação de desenvolvimento de software, incluindo artefatos, arquiteturas, ferramentas entre outros. Serão abordados dois principais aspectos: primeiramente, a geração automática de documentação dentro do processo de desenvolvimento de software e depois a geração de outros artefatos deste processo, a partir das definições de alto nível disponíveis através do *framework.*

Exemplos de aplicações do *FrameworkDoc* em projetos reais são apresentados. No entanto, os documentos e artefatos de software considerados foram definidos de forma suficientemente genérica para serem aproveitados em outros contextos.

Palavras-Chave: Engenharia de Software, *Frameworks*, Documentação de Software, Geração de Artefatos.

FrameworkDoc: A Documentation and Artifact Generation Tool

ABSTRACT

Today, one of the great challenges for any software developer is to design systems that reuse the maximum code elements of existing projects in order to reduce time and effort during software production. Among the several forms of reuse we highlight the framework approach, in the context of object-oriented software development.

The importance of documentation used in the software development process allied to the *framework* features were the motivations for the accomplishment of the present study. In a process of software development, documentation is not part of a very well defined phase, it takes place all over the process, in parallel with other phases of the life cycle. The framework approach within this context focuses on the management of *templates* and definitions of features of software artifacts (comprising not only the code but also analysis products, project, *frameworks*, components, diagrams, among others), making the process easier and accelerating documentation.

Due to its unique features, which are examined and demonstrated in the present study, a framework contains a series of information that can be useful for the production of other artifacts of the development process, such as test planning, database scripts, code standards, among others. Thus, in a process of evolutionary development, which uses the artifact generation as resource, maintenance can be fully accomplished through specification, and not distributed throughout generated artifacts.

The goal of the present study is to investigate, propose and develop a software documentation and artifact generation tool named FrameworkDoc. The term software documentation we use refers to the documentation of software development, including artifacts, architectures, and tools, among others. Two major aspects will be approached: firstly, the automatic generation of documentation within the software development process and the generation of other artifacts of this process, based on high-level definitions available through the *framework*. AApplications in real projects were also included.

Keywords: Software engineering, *Frameworks*, Software documentation, Artifacts Generation.

1 INTRODUÇÃO

A Engenharia de Software tem como objetivo viabilizar maior produtividade na construção de aplicações e qualidade dos artefatos de software (GHEZZI; JAZAYERI; MANDRIOLI, 1991; PRESSMAN 1995).

Diversas foram as metodologias e processos que surgiram com tais perspectivas, fornecendo descrições de como o software deveria ser criado e mantido (ORTIGOSA, 1995). De uma forma geral, estas descrições envolvem uma contínua produção e transformação de notações, desde a especificação de requisitos até o código do software produzido.

A grande preocupação com a qualidade dos softwares produzidos, a redução de tempo de desenvolvimento e a redução dos custos de projeto são as métricas mais estudadas dentro das fábricas de software (GOMES; OLIVEIRA; ROCHA, 2001). Isto porque a indústria do software teve natural evolução devido às exigências do mercado.

Associadas as métricas, surgiram métodos, técnicas e paradigmas que possibilitem atingi-las de forma rápida e eficiente.

Com o surgimento da Orientação a Objetos (OO), apareceram inúmeras metodologias de desenvolvimento de software, com o objetivo de permitir a produção de software OO e, ao mesmo tempo, aumentar a produtividade de desenvolvimento (FURLAN, 1998). Atualmente a UML - *Unified Modeling Language* (RUMBAUGH; JACOBSON; BOOCH, 1999; BOOCH; RUMBAUGH; JACOBSON, 2000) é a linguagem de modelagem de softwares OO padrão com larga utilização na indústria. Juntamente com esta linguagem de modelagem, surgiram inúmeros processos de desenvolvimento com suporte à UML, onde o *RUP – Rational Unified Process* (JACOBSON; RUMBAUGH; BOOCH, 1999), é ótimo guia para o uso ideal da UML (MATOS, 2002), centrado na gerência de riscos e na modelagem da arquitetura do software, e que serviu de base para criação de outros processos de desenvolvimento, com a adição de novos artefatos e iterações.

Atualmente, um dos grandes desafios para qualquer desenvolvedor de software é projetar um sistema que reutilize ao máximo código e projetos existentes (TREVISAN, 1994). Dentro deste contexto, tem-se a abordagem de *frameworks.*

Frameworks[1] são estruturas que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de software. A grande vantagem

[1] A expressão *frameworks* abordada neste trabalho se refere a estruturas inacabadas, desenvolvidas para um determinado propósito, abrangendo também estrutura de classes.

desta abordagem é a promoção de reuso de código e projeto, que diminui o tempo e o esforço exigidos na produção do software (SILVA, 2000).

Ao mesmo tempo, a integração de todos os artefatos de software utilizados no processo de desenvolvimento bem como a disponibilização destes de forma organizada à equipe possibilitam um trabalho planejado, eficiente e de qualidade. Isto se deve ao fato de que a qualquer momento do processo de desenvolvimento pode-se obter o andamento e acompanhamento do projeto.

A grande importância da documentação de software utilizada no processo de desenvolvimento aliada às características de *frameworks* serviram como motivação para a realização deste trabalho.

Em particular, este trabalho surgiu da necessidade de criação de uma ferramenta de documentação e geração de artefatos de software, contemplando tanto o PADS - Processo APOENA[2] de desenvolvimento Software, quanto outros artefatos para desenvolvimento e uso do software.

No entanto, tanto a ferramenta proposta quanto os artefatos de software considerados neste trabalho são suficientemente genéricos para serem aproveitados em outros contextos.

A integração da abordagem de *frameworks* à documentação de software é muito importante pelo fato de aumentar a produtividade de projeto e código, além de fornecer mecanismos para a geração de artefatos.

Neste caso, serão abordados dois principais aspectos relativos a integração de *frameworks* e documentação de software: primeiramente, a documentação dentro do processo de desenvolvimento de software e, depois, devido as próprias características dos *frameworks*, a geração de artefatos a partir de definições das especificações de alto nível.

A documentação dentro do processo de desenvolvimento de software não faz parte de uma fase definida, mas ocorre durante toda sua existência, em paralelo com outras fases do ciclo de vida. Ela pode ser definida de forma linear ou associativa (BORGES, 1998).

A abordagem realizada de forma linear é seqüencial, onde todo e qualquer documento é desenvolvido para ser lido em uma ordem pré-determinada e inalterável. Desta forma, o engenheiro de software está submetido a um nível de abstração pré-determinado, não tendo a opção de aprofundamento do conteúdo relacionado ao tema. A documentação terá como característica uma disjunção entre os documentos gerados.

O suporte oferecido por uma abordagem associativa, que é baseada em hipertexto, é semelhante a forma linear, porém com o documento contendo referências à

---

[2] APOENA Software Livre é uma *software-house* que trabalha com desenvolvimento de software livre e *open-source*, local de trabalho do autor. A documentação utilizada no processo de desenvolvimento juntamente com o código-fonte são componentes do produto entregue ao cliente. Mais informações em http://www.apoenasoftwarelivre.com.br

documentação de outros artefatos gerados na mesma ou em outra fase do processo de desenvolvimento de software.

Em (BORGES, 1998), tem-se algumas vantagens dos hipertextos em relação a textos lineares:

- Proporcionam conectividade entre as informações;
- Oferecem uma interface compatível com o modo do raciocínio humano;
- Permite ao usuário definir o nível de abstração desejado, navegando entre a documentação dos artefatos.

Como as etapas de desenvolvimento de software são inter-relacionadas, os textos lineares apresentam uma deficiência: a falta de capacidade de expressar a fronteira deste relacionamento. Portanto, se torna difícil a utilização de um único texto linear como documentação de todo o processo, não permitindo a visualização individual de cada etapa, dificultando a localização da informação e por outro lado utilizando vários textos lineares com a documentação de cada etapa, não fornecendo uma visão global do processo.

A abordagem de *frameworks* dentro deste contexto enfoca o tratamento de *templates* e definições das características dos artefatos, facilitando e acelerando o processo de documentação.

Para a geração de artefatos, os *frameworks* possuem características semelhantes aos geradores de artefatos (FRANCA; STAA, 2001). Um gerador de artefatos é uma ferramenta de software que produz um artefato a partir de uma especificação de alto nível.

Um processo de desenvolvimento evolutivo, que utiliza a geração de artefatos como recurso, preconiza que a manutenção seja sempre e integralmente realizada na sua especificação e não nos artefatos gerados. Cada ciclo produz uma versão executável do artefato.

As similaridades encontradas entre *frameworks* e geradores de artefatos, sob a ótica de *frameworks* estão relacionadas as partes fixas[3] e variáveis[4] do artefato gerado. A atividade de configuração de *hot-spots* de um *framework* corresponde ao fornecimento da especificação do artefato que vai ser gerado. A instanciação do *framework*, no contexto do gerador, corresponde aos mecanismos de geração. Relacionado a implementação, os *frameworks* utilizam mecanismos OO, e os geradores utilizam outros mecanismos (FILETO; MEIRA; COSTA; MASSHURÁ, 1996).

1.1 Objetivos

O objetivo principal deste trabalho é investigar, propor e desenvolver uma ferramenta para documentação e geração de artefatos de software, denominado *FrameworkDoc*, a partir de estudos relacionados ao uso de ferramentas, tecnologias, *frameworks*, documentação de software e geração de artefatos no processo de desenvolvimento de software.

---

[3] *frozen-spots*

[4] *hot-spots*

Como objetivos secundários, tem-se:

- Estudar os conceitos, características, construção e uso de *frameworks;*
- Realizar um estudo dos artefatos de software genéricos presentes em vários processos de desenvolvimento de software;
- Investigar as formas de integração e geração de artefatos junto a *frameworks*;
- Realizar engenharia de requisitos, análise, projeto e implementação do *FrameworkDoc*;
- Realizar estudo(s) de caso do *FrameworkDoc* para validação da ferramenta e de sua aplicabilidade.

O trabalho está organizado da seguinte forma:

No capítulo 2, é abordada uma visão panorâmica relacionada à *frameworks*, documentação e processos de desenvolvimento de software e integração de *frameworks* à documentação de software. Inicialmente, apresenta um estudo relacionado à *frameworks*, incluindo: conceitos, características, classificação dos *frameworks*, arquitetura e componentes de um *framework* e outras abordagens de reutilização. Relacionado a documentação e processos de desenvolvimento de software, é apresentado um estudo sobre processos incrementais e iterativos, relacionando estrutura, fases e artefatos sobre o *RUP – Rational Unified Process*, *XP - eXtreme Programming*, PMR - Processo e Modelos Recomendados de *Craig Larman* e o PADS – Processo APOENA de Desenvolvimento de Software. É relacionado também a abordagem ágil nos processos de desenvolvimento. E, por último, é abordado a forma de integração de *frameworks* a documentação de software.

No capítulo 3, é apresentado a ferramenta *FrameworkDoc*, definindo sua estrutura e tecnologias utilizadas para sua implementação, além dos passos para configuração de projetos. Quanto a estrutura, são apresentados os componentes e os principais artefatos documentados e gerados pela ferramenta. Relacionado as tecnologias, são apresentadas as tecnologias utilizadas para desenvolvimento do *FrameworkDoc*. No projeto, é apresentada a arquitetura MVC (*Model-View-Controller*) com seus componentes e padrões implementados.

No capítulo 4, são apresentados os estudos de caso para validação da ferramenta. Neste estudo de caso, são relacionados três projetos específicos, com características distintas, onde é validada a ferramenta.

No capítulo 5 é apresentado as considerações finais deste trabalho, contribuições e trabalhos futuros que poderão surgir a partir deste.

2 FRAMEWORKS, DOCUMENTAÇÃO E PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE E SUA INTEGRAÇÃO: UMA VISÃO PANORÂMICA

Neste capítulo é apresentado um estudo sobre *frameworks*, documentação e processos de desenvolvimento de software. Inicialmente, apresenta uma introdução ao estudo de *frameworks*, abordando conceitos, características, classificação dos *frameworks*, arquitetura e componentes de um *framework* e outras abordagens de reutilização. Após, tem-se um estudo sobre documentação e processos de desenvolvimento de software, onde são apresentados alguns processos incrementais e iterativos, que utilizam a UML como linguagem de modelagem, relacionando estrutura, fases e artefatos sobre o RUP, XP, PMR de *Craig Larman* e o PADS. É relacionado também a abordagem ágil nos processos de desenvolvimento. Dentro deste contexto, ainda tem-se um estudo sobre a integração de *frameworks* a documentação de software.

2.1 *Frameworks*

Existem muitas formas de se promover a reutilização em desenvolvimento de software, desde padrões e herança até o código (AMBLER, 1998). Dentro deste contexto, os *frameworks* são uma abordagem de projeto e código em um nível de granularidade elevado.

Os *frameworks* possibilitam aumentar a produtividade e a qualidade no desenvolvimento de aplicações. O desenvolvimento parte de uma aplicação pré-implementada, a qual é desenvolvida por projetistas experientes em um domínio de aplicação (JOHNSON, 1992).

2.1.1 Definições Gerais

Em (CRESPO, 2000), são apresentadas algumas definições de *frameworks*:

"*Um framework é um conjunto de objetos que colaboram entre si, com o objetivo de atender um conjunto de responsabilidades para uma aplicação específica em um dado domínio.*" (JOHNSON; RUSSO, 1991; GAMMA; HELM; JOHNSON; VLISSIDES, 2000).

"*Um framework é uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial para especialização.*" (MATTSSON, 2000).

"*um software parcialmente completo, projetado para ser instanciado. São definidos uma arquitetura para a família de subsistemas, oferecendo os construtores básicos*

para criá-los e definindo os pontos de adaptação do código para um funcionamento específico dos módulos." (BUSCHMANN; MEUNIER; ROHNERT; SOMMERLAD; STAL, 1996; PREE, 1995).

Um *framework,* como já definido por vários autores, é uma aplicação semi-acabada que, muitas vezes pode ser um sistema inteiro ou, às vezes, um subsistema, sempre vinculado ao paradigma da OO. Mais recentemente, o termo passou a ser mais abrangente, significando que um *framework* é qualquer solução incompleta que pode ser completada através da instanciação[5] e, desta forma, possibilitando a geração de mais de uma aplicação dentro do domínio-alvo do *framework* (FONTOURA, 1999; FRANCA, 2001).

2.1.2 Características

Os *frameworks* apresentam inúmeras características, sendo que se destacam duas delas: inversão do controle e fornecimento de infra-estrutura e projeto.

A inversão de controle acontece porque os engenheiros de software reutilizam, na maioria das vezes, componentes de uma biblioteca em que o programa principal chama os componentes quando necessário, decidindo quando o componente será chamado e suas interações com os demais componentes. Quando se utiliza a abordagem de *frameworks*, o programa principal é reutilizado e o engenheiro de software decide o que será conectado dentro dele, determinando a estrutura geral e o fluxo de controle dos programas (FAYAD; SCHMIDT; JOHNSON, 1999).

Os *frameworks* fornecem infra-estrutura de projeto ao engenheiro de software, reduzindo assim a quantidade de código a ser desenvolvido, testado e depurado com ele. É definida a arquitetura da aplicação (SILVA, 2000).

Um *framework* se destina a gerar diferentes aplicações para um domínio, onde podem ser identificados inúmeros benefícios (FAYAD; SCHMIDT; JOHNSON, 1999; CRESPO, 2000):

- Modularidade: *frameworks* aumentam a modularidade, encapsulando detalhes de implementação sob interfaces bem definidas e estáveis. Esta modularidade auxilia na qualidade do software, localizando os lugares de impactos no *design* e nas trocas de implementações, reduzindo o esforço necessário para entender o *design* e para realizar futuras manutenções;
- Reutilização: interfaces estáveis presentes nos *frameworks* aumentam o potencial de reutilização pela definição de componentes abstratos que podem ser redefinidos para criarem novas aplicações. O aproveitamento dos componentes já definidos aumentam a produtividade dos engenheiros de software, aumentando por sua vez a qualidade, desempenho e confiabilidade do software;

[5] A atividade de se construir uma aplicação através de um *framework*, é também conhecida como *instanciação do framework* (ZANCAN, 1999).

- Extensibilidade: um *framework* aumenta a extensibilidade na medida em que oferece os métodos *hook*[6] explícitos, permitindo a extensão das aplicações através de interfaces estáveis presentes.

2.1.3 Classificação dos *Frameworks*

Os *frameworks* podem ser caracterizados por diferentes dimensões. As classificações de maior relevância são: quanto ao domínio do problema, quanto à estrutura e quanto ao uso (CRESPO, 2000).

Relacionado ao domínio do problema, existem *frameworks* aplicados a diferentes domínios de problema, como interface gráfica com usuário, persistência, multimídia, acesso a dispositivos, entre outros (WEINAND; GAMMA; MARTY, 1994).

Relacionado à estrutura interna, torna-se fácil o entendimento do seu comportamento, relacionada a concepção da arquitetura do software (BUSCHMANN, 1994). As principais arquiteturas de *frameworks* são em camadas (diferentes níveis de abstração), MVC (separa a interface do núcleo de funcionalidade), arquitetura reflexiva (considera adaptações futuras de tecnologia, ambiente e requisitos, sem alteração na estrutura), *brokers* (utilizada em sistemas distribuídos, através de chamadas remotas), entre outras.

Relacionado ao uso do *framework*[7], tem-se dois aspectos: *Caixa branca* (utilização através da escrita de subclasses, predominantemente) e *Caixa preta* (através da composição e conexão de componentes).

2.1.4 Arquitetura e Componentes de um *Framework*

Um *framework* possui um *kernel* que contém um conjunto de classes que não são passíveis de adaptação e que permanecem presentes em todas as suas instanciações (CRESPO, 2000).

A diferenciação das aplicações que instanciam um mesmo *framework* são as adaptações realizadas nos pontos de flexibilização (*hot-spots*).

Os *hot-spots* dirigem a especialização do *framework*, podendo ser feita tanto por herança como delegação. Os *frozen-spots* são as partes fixas do *framework.*

A figura 2.1 ilustra a arquitetura e os componentes de um *framework*

---

[6] Os métodos *hook* possuem implementação inicial a qual pode ser ou não redefinida em subclasses. Mais informações sobre tipos de métodos de classes abstratas podem ser encontrados em (WIRFS-BROCK; JOHNSON, 1990) e (JOHNSON; RUSSO, 1991).

[7] *Frameworks White-Box* se caracterizam pela necessidade de conhecimento de suas estruturas de classes e protocolos de colaboração para a construção de aplicações. *Frameworks Black-Box* são utilizados sem a necessidade da compreensão detalhada de estruturas de classes e protocolos de colaboração (JOHNSON; FOOTE, 1988).

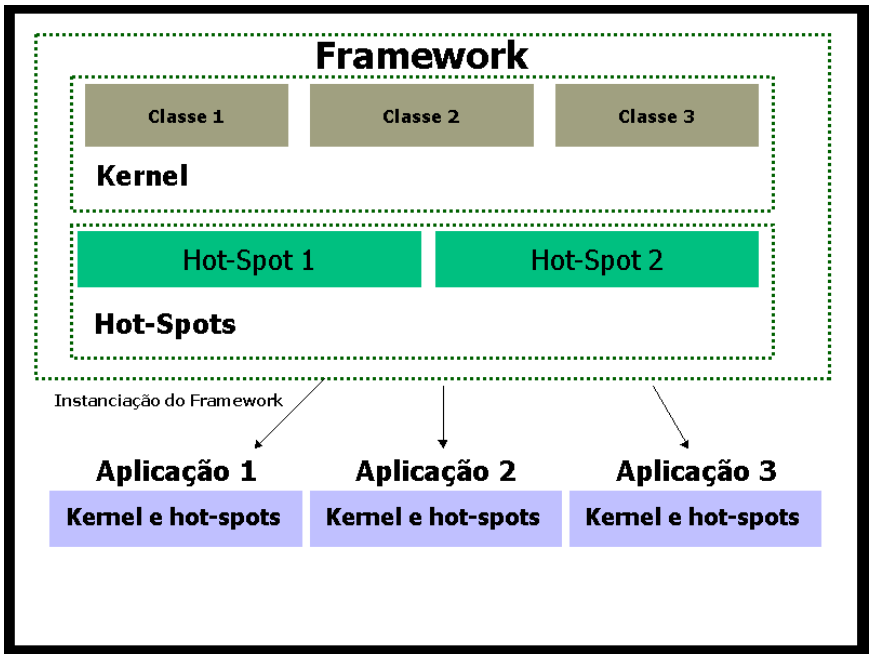

Figura 2.1: Arquitetura e Componentes de um *Framework*

2.1.5 *Frameworks* e outras abordagens

Os *frameworks* não são a única forma de reutilização. Existem outras formas de reutilização que são, segundo (CRESPO, 2000): Padrões de projeto (GAMMA; HELM; JOHNSON; VLISSIDES, 2000), linguagens de padrões (BRAGA; MASIERO, 2001), bibliotecas de classes, aplicações orientadas a objetos e *frameworks* de arquitetura (CRESPO, 2000).

2.2 Documentação e Processos de Desenvolvimento de Software

Um processo é um conjunto de passos parcialmente ordenados com a intenção de atingir uma meta. Na Engenharia de Software, um processo ou ciclo de vida baseia-se na fundamentação de várias atividades. Tais atividades identificam ações bem definidas, constantemente associadas à análise e especificação de requisitos, projeto, implementação, implantação e testes (MATOS, 2002). Sua meta é entregar, de maneira eficiente e previsível, um produto de software capaz de atender às necessidades do cliente (BOOCH; RUMBAUGH; JACOBSON, 2000).

Historicamente, os processos de desenvolvimento eram baseados em ciclos seqüenciais onde cada etapa dependia totalmente do término da etapa anterior, acarretando, dentre outros problemas, o desânimo e impaciência do cliente.

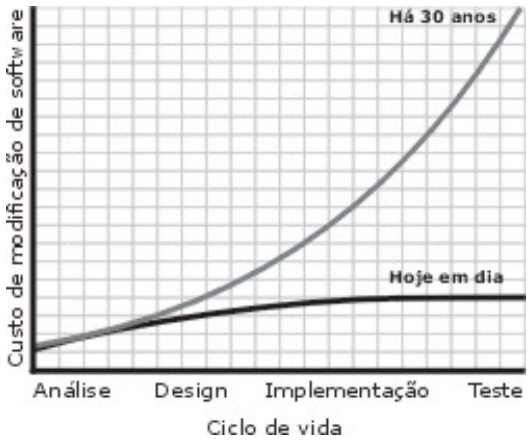

Figura 2.2: Ciclo de vida do Software X Custo de modificação (WUESTEFELD, 2004)

Processos de desenvolvimento de software, principalmente ligados à orientação a objetos, devem ser iterativos, permitindo a avaliação dos resultados parciais e a verificação dos riscos (MATOS, 2002).

A UML é amplamente independente de processo, o que a torna passível de utilização por outros processos de desenvolvimento (FOWLER; SCOTT, 2000). É uma linguagem destinada a visualização, especificação, construção e documentação de artefatos de software, permitindo a visualização da aplicação sob várias perspectivas.

Nesta seção são apresentados os processos de desenvolvimento de software e boas práticas que serviram como base para a criação do *PADS – Processo APOENA de Desenvolvimento de Software*.

2.2.1 Rational Unified Process (RUP)

O *Rational Unified Process* (RUP) é um processo de desenvolvimento de software, cujas principais características são a orientação por casos de uso, centralizado na arquitetura, sendo um processo iterativo e incremental (JACOBSON; RUMBAUGH; BOOCH , 1999).

O RUP surgiu da evolução do *Objectory* (FURLAN, 1998), podendo ser decomposto em fases. Uma fase é o intervalo de tempo decorrido entre dois importantes pontos do processo, quando um conjunto bem definido de objetivos é alcançado, os artefatos são concluídos e decisões são tomadas para se passar à fase seguinte.

As atividades do RUP dão ênfase à criação e manutenção de modelos no lugar de documentos impressos. Essas representações podem ser visualizadas de várias maneiras e as informações representadas podem ser capturadas instantaneamente e controladas eletronicamente.

O RUP encoraja o controle de qualidade e gerenciamento de riscos, contínuos e objetivos (BOOCH; RUMBAUGH; JACOBSON, 2000). A avaliação da qualidade é inserida no processo, em todas as atividades e envolvendo todos os participantes, com a utilização de métricas e objetivos. O gerenciamento de riscos é inserido no processo, de forma que são identificados e atacados no início do processo de desenvolvimento, onde há tempo para uma reação.

2.2.1.1 Fases e Iterações

Em cada fase, ocorrem várias iterações. Uma iteração representa um ciclo completo de desenvolvimento, desde a elicitação dos requisitos na análise até a implementação e a realização dos testes, resultando na versão de produto executável.

Existem quatro fases no ciclo de desenvolvimento de software definido pela RUP (BOOCH; RUMBAUGH; JACOBSON, 2000):

- Concepção (*Inception*): é a primeira fase do processo, onde a idéia inicial para o desenvolvimento é levada até o ponto de ser suficientemente fundamentada para assegurar à fase de elaboração. É a fase em que os requisitos são identificados e avaliados;
- Elaboração (*Elaboration*): Fase onde são definidas a visão do produto e a arquitetura do software. A arquitetura permite uma avaliação do negócio como um todo, muito importante para o ambiente de software. Nesta fase, os requisitos são articulados e são definidas as prioridades;
- Construção (*Construction*): Fase onde o software chega a um formato executável. Os requisitos e principalmente os critérios de avaliação são constantemente reexaminados em relação às necessidades comerciais do projeto e os recursos são alocados de modo adequado a atacar ativamente os riscos do projeto;
- Transição (*Transition*): É a quarta fase do processo, em que o software chega às mãos da comunidade de usuários. Após a disponibilização do software à comunidade, sempre surgem questões que requerem algum desenvolvimento adicional, com a finalidade de ajustar o software. Nesta fase são avaliados os objetivos do projeto, determinando a iniciação de outro ciclo de desenvolvimento.

A figura 2.3 apresenta o ciclo de vida de desenvolvimento de software, definido pelo RUP (BOOCH; RUMBAUGH; JACOBSON, 2000).

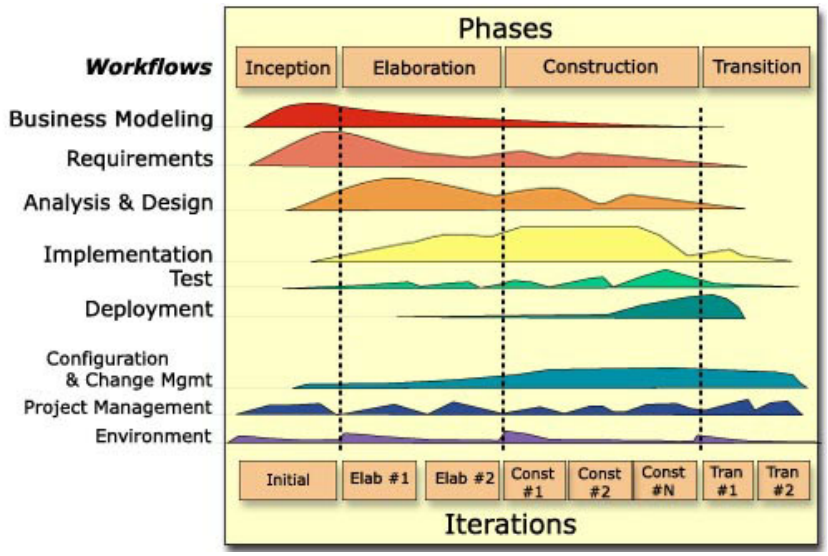

Figura 2.3: *Framework* RUP

A concepção e a elaboração abrangem atividades de engenharia do ciclo de vida do desenvolvimento. A construção e a transição constituem a produção do software.

2.2.1.2 Ciclos de Desenvolvimento

A passagem pelas quatro principais fases é chamada de ciclo de desenvolvimento, resultando em uma versão do executável (BOOCH; RUMBAUGH; JACOBSON, 2000). O primeiro passo das quatro fases é chamado de ciclo inicial de desenvolvimento. Um produto existente evoluirá para sua próxima geração pela repetição da mesma seqüência de fases de concepção, elaboração, construção e transição. Estes ciclos proporcionarão a evolução do software e são conhecidos como ciclos de evolução.

O RUP possui nove fluxos de trabalho (BOOCH; RUMBAUGH; JACOBSON, 2000; MATOS, 2002):

- Modelagem do Negócio (*Business Modeling*): Descreve a estrutura e a dinâmica da empresa, representando todo o ambiente de execução do cliente;
- Requisitos (*Requirements*): Descreve o método baseado em casos de uso para identificar requisitos, onde estes são constantemente verificados;
- Análise e Projeto (*Analysis & Design*): Descreve as várias visões da arquitetura. A análise somente é iniciada quando o negócio estiver sido amplamente verificado;
- Implementação (*Implementation*): Mais forte constatação de que o processo é iterativo. Pode-se perceber que a implementação tem início em conjunto com a análise, por exemplo, por meio da construção do primeiro protótipo. Leva-se em consideração o desenvolvimento do software, testes de unidade e integração;
- Testes (*Tests*): Descreve os casos de teste, procedimentos e medidas para acompanhamento de erros;
- Entrega (*Deployment*): Abrange a configuração do sistema a ser entregue;
- Gerenciamento de Configurações (*Configuration & Change Mgmt*): Controla as modificações dos artefatos do projeto;
- Gerenciamento de Projeto (*Project Management*): Apresenta várias estratégias para o trabalho com um processo iterativo;
- Ambiente (*Environment*): Abrange a infra-estrutura necessária para o desenvolvimento do software.

Em cada fluxo de trabalho, são capturados um conjunto de atividades e artefatos correlacionados. Há conexões importantes entre os artefatos que estão associados entre os *workflows*. O modelo de caso de uso gerado durante a elicitação dos requisitos é realizado pelo modelo de projeto a partir do processo de análise e projeto, codificado pelo modelo de implementação a partir do processo de implementação e verificado pelo modelo de teste a partir do processo de teste.

Na tabela 2.1, tem-se os modelos da RUP (BOOCH; RUMBAUGH; JACOBSON, 2000).

Tabela 2.1: Modelos do RUP

Modelo	Descrição
Modelo de Negócio	Estabelece a abstração da empresa.
Modelo de Domínio	Estabelece o contexto do sistema.
Modelo de Caso de Uso	Estabelece os requisitos funcionais do sistema.
Modelo de Análise (opcional)	Estabelece o projeto de uma idéia, focando a possível solução.
Modelo de Projeto	Estabelece o vocabulário do problema e de sua solução.
Modelo de Processo (opcional)	Estabelece mecanismos de concorrência e sincronização do sistema.
Modelo de Implantação	Estabelece a topologia do hardware em que o sistema é executado.
Modelo de Implementação	Estabelece as partes utilizadas para montar e liberar o produto de software.
Modelo de Teste	Estabelece os caminhos pelos quais o sistema é verificado e validado.

Uma visão é uma projeção em modelo. No RUP, a arquitetura é capturada em cinco visões interligadas: visão de projeto (vocabulário, funcionalidades), visão de processo (desempenho, escalabilidade), visão de implantação (topologia do software, distribuição, instalação), visão de implementação (construção do software) e visão de caso de uso (comportamento).

Os artefatos do RUP são categorizados como artefatos de gerenciamento ou artefatos técnicos (BOOCH; RUMBAUGH; JACOBSON, 2000). Na tabela 2.2, tem-se a divisão dos artefatos técnicos em quatro conjuntos principais.

O RUP é um processo configurável (BOOCH; RUMBAUGH; JACOBSON, 2000). Embora não exista um único processo adequado para todas as empresas de desenvolvimento de software, o RUP pode ser ajustado e redimensionado para atender às necessidades de projetos que variam desde pequenas equipes até grandes empresas de desenvolvimento de software. É fundamentado em uma arquitetura de processo simples e claro, proporcionando uma base comum em um conjunto de processos e ainda pode ser modificada para acomodar várias situações.

Pela sua característica de adaptabilidade, o RUP descreve mais de 100 artefatos que podem ser utilizados dentro do processo de desenvolvimento, onde os documentos são formados por textos e diagramas, agrupados em formato eletrônico (SMITH, 2004).

Muitas organizações utilizam seu próprio processo de desenvolvimento de software, baseado no RUP, obtendo resultados altamente positivos (MATOS, 2002).

Tabela 2.2: Agrupamento de Artefatos da RUP

Conjunto	Descrição
Conjunto de Requisitos	Agrupa as informações descrevendo a funcionalidade do software.
Conjunto de Projeto	Agrupa informações descrevendo como o software será construído, capturando decisões, levando em conta tempo, orçamento, reutilização, objetivos de qualidade entre outras métricas.
Conjunto de Implementação	Agrupa informações sobre os elementos de software que compõe o produto.
Conjunto de Implantação	Agrupa as informações sobre a forma de como o software será empacotado, instalado e executado no ambiente de destino.

2.2.2 Processo UML de Larman

O processo de desenvolvimento de software tem por finalidade organizar as atividades relacionadas a criação, entrega e manutenção de software. O processo definido por Craig Larman (LARMAN, 2003) fornece uma fundamentação para criar um processo de desenvolvimento administrável, repetível e bem sucedido, conhecido como *Processo e Modelos Recomendados – PMR* (LARMAN, 2000).

O PMR apresenta as melhores práticas, das atividades básicas e dos modelos empregados, em processo de desenvolvimento OO baseado em uma abordagem iterativa e incremental, dirigida por casos de uso (OBJECT DESIGN, 2003). Os passos e roteiros definidos em (LARMAN, 2000) são um roteiro através do desenvolvimento de software, utilizando a tecnologia de objetos.

O PMR é uma descrição de processos e artefatos baseados em vários outros métodos como *Fusion* (COLEMAN, 1994), *Martin-Odell* (MARTIN; ODELL, 1995), OOSE – Objectory (JACOBSON, 1992), OMT (RUMBAUGH, 1991) e *Booch* (BOOCH, 1994).

2.2.2.1 Estrutura do Processo

O processo, em um nível maior de abstração, incluem as seguintes fases:

- Planejar e Elaborar: Engloba planejamento, definição de requisitos, construção de protótipos;
- Construir: Engloba refinar plano, sincronizar artefatos, análise, projeto, implementação e testes;
- Instalar: Engloba a implantação do software para uso.

O desenvolvimento é iterativo, baseado no aumento e no refinamento sucessivo do software através de múltiplos ciclos de análise, projeto, implementação e testes (LARMAN, 2000). Isto possibilita vantagens de controle da complexidade e o rápido *feedback* do processo, com implementações de pequenos subconjuntos do software.

O processo é focado nos requisitos principais, onde são definidos os casos de uso e suas prioridades. Neste caso, alguns ciclos de desenvolvimento, especialmente os iniciais, devem ser focados em requisitos não evidentes, como funções de serviço (LARMAN, 2000).

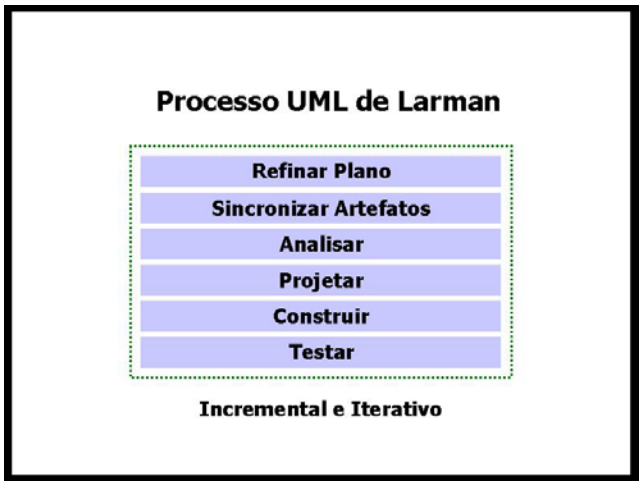

Figura 2.4: Processo UML de Larman

2.2.2.2 Fase Planejar e Elaborar

O processo, em um nível maior de abstração, incluem as seguintes fases:

Nesta fase, são abordados aspectos relacionados a concepção inicial do software, investigando alternativas, definindo o planejamento e a especificação dos requisitos.

Na tabela 2.3, tem-se a descrição dos artefatos desta fase (LARMAN, 2000).

Tabela 2.3: Artefatos da Fase Elaborar e Planejar

Artefato	Descrição
Plano	Inclui cronograma, recursos, orçamento.
Relatório de Investigação Preliminar	Engloba motivação, alternativas, necessidades de negócio.
Especificação dos Requisitos	Aborda os requisitos de forma expositiva.
Glossário	É um dicionário de termos e outras informações relacionadas ao contexto do software. Se mantém durante todo o processo de desenvolvimento.
Protótipo	Utilitário gerado para dar maior compreensão do problema.
Casos de Uso Preliminares	Descrições dos processos do domínio.
Diagrama de Casos de Uso	Ilustra o relacionamento entre os casos de uso.
Rascunho do Modelo Conceitual	É um modelo conceitual preliminar, auxiliando na compreensão do vocabulário do domínio, como eles se relacionam com os casos de uso e a especificação dos requisitos.

2.2.2.3 Fase Construir

Nesta fase, o projeto passa por repetidos ciclos de desenvolvimento em definidas em janela de tempo[8], dentro dos quais o software é ampliado. O objetivo final é um software em operação que atenda corretamente os requisitos (LARMAN, 2000).

Na tabela 2.4, tem-se a descrição dos artefatos desta fase.

Tabela 2.4: Artefatos da Fase Construir

Etapa	Artefato	Descrição
Analisar	Casos de Uso Essenciais	Permite a compreensão dos processos e requisitos, através de uma narrativa do processo.
	Diagrama de Casos de Uso	Ilustra o relacionamento entre os casos de uso e os atores.
	Modelo Conceitual	Define a estrutura de classes e atributos, contendo o relacionamento entre estas classes.
	Diagramas de Seqüência do Sistema	Define-se os diagramas de seqüência do sistema baseado nos casos de uso essenciais.
	Contratos de Operação	Define a estrutura de um método projetado, através de pré-condições, algoritmo, pós-condições, exceções.
	Diagramas de Estado	Aborda o estado dos objetos e os eventos necessários para que ocorra a mudança.
Projetar	Casos de Uso Reais	Descreve o projeto real do caso de uso em termos da tecnologia concreta de entrada e de saída.
	Diagrama de Classes de Projeto	Define a expansão do modelo conceitual, com a integração das classes de serviço.

[8] Tempo fixado para uma iteração

Etapa	Artefato	Descrição
Projetar	Diagramas de Interação	Define os diagramas de seqüência e colaboração das classes definidas.
	Definição de Padrões	São definidos os padrões de arquitetura e de projeto que serão utilizados.
	Diagrama de Pacotes da Arquitetura	Os modelos são estruturados de acordo com o seu relacionamento semântico
	Definição do Esquema de Banco de Dados	É definido o esquema de Banco de Dados, conforme modelo conceitual
Construir	Definições de Classes/Interfaces e Métodos	Definição e criação dos componentes do software.
	Criação da GUI	Criação da Interface gráfica, baseada no protótipo e nos casos de uso reais.
Testar	Casos de Teste	Através dos casos de uso essenciais e reais são projetados os testes de unidade, integração e aceitação.

Ainda, segundo (LARMAN, 2000), alguns artefatos pode ser criados paralelamente. Outros, são dependentes. Na figura 2.5, tem-se a dependência entre os artefatos definidos pelo PMR.

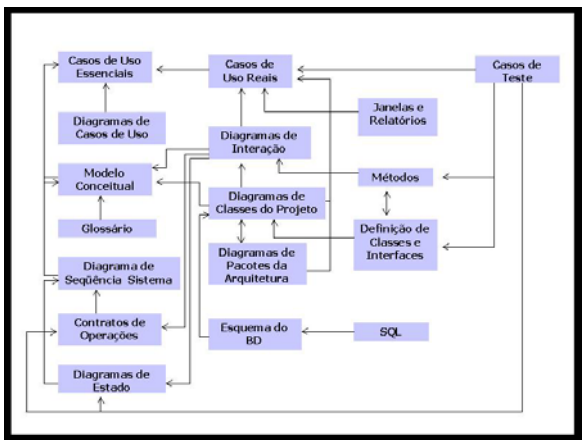

Figura 2.5: Dependência dos Artefatos

2.2.3 Modelagem Ágil (MA)

Segundo (AMBLER, 2004) a modelagem ágil (MA) é uma metodologia baseada na prática para criação de modelos efetivos e a documentação de software. É baseada em um conjunto de valores, princípios e práticas para a criação de modelos, aplicados em projetos de desenvolvimento de software de forma efetiva e leve.

A modelagem ágil adotou alguns valores definidos na Aliança Ágil[9], onde alguns dos desenvolvedores mais conhecidos da indústria definiram um manifesto[10], relacionado a seguir:

> "Estamos evidenciando maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:
>
> 1. Indivíduos e interação **MAIS QUE** processos e ferramentas;
>
> 2. Software em funcionamento **MAIS QUE** documentação abrangente;
>
> 3. Colaboração com o cliente **MAIS QUE** negociação de contratos;
>
> 4. Responder a mudanças **MAIS QUE** seguir um plano.
>
> *Ou seja, mesmo tendo valor os itens à direita, valorizamos mais os itens à esquerda."*
>
> Autores: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

A MA possui três objetivos:

- Definir e mostrar como colocar em prática uma coleção de valores, princípios e práticas pertinentes à modelagem efetiva;
- Explorar como aplicar técnicas de modelagem em projetos de software através de uma abordagem ágil como XP, DSDM ou SCRUM[11];

---

[9] Mais informações em http://www.agilealliance.org

[10] Mais informações em http://agilemanifesto.org

[11] Neste trabalho é abordado uso e práticas do XP, que é uma disciplina ágil. Mais informações sobre métodos ágeis em http://www.agilealliance.org/programs/roadmaps/Roadmap/index.htm

- Explorar como melhorar a modelagem sob processos prescritivos como RUP

2.2.3.1 Valores da MA

Os valores definidos na MA, conforme (AMBLER, 2004) são:

- Comunicação: a comunicação deve ser efetiva entre todos os envolvidos no projeto.
- Coragem: eventualmente se precisa de grandes *refactorings* no modelo. É necessário tomar grandes decisões e seguí-las.
- *Feedback*: é imperativo para se ter certeza de que está fazendo a coisa certa ou para fazer uma correção assim que possível.
- Humildade: trabalhar com as pessoas, valorizar a comunicação, a fim de eliminar possíveis erros e ter outros pontos de vista sobre os modelos.
- Simplicidade: fazer a coisa mais simples que possa funcionar.

2.2.3.2 Princípios e Práticas da MA

Os princípios e práticas da MA estão divididas em centrais e suplementares (AMBLER, 2004). Na tabela 2.5, tem –se a listagem dos princípios e práticas.

Tabela 2.5: Princípios e Práticas da MA

Descrição	Centrais	Suplementares
Princípios	• Simplicidade assumida • Adotar mudanças • Capacitar o próximo esforço é seu objetivo secundário • Mudanças incrementais • Maximizar o investimento daqueles que suportam o sistema • Modelar com propósito • Usar múltiplos modelos • Trabalho de qualidade • *Feedback* rápido • Software é o objetivo primário • Um modelo vale mais que 1024 linhas de código	• Conteúdo é mais importante que representação • Todos aprendem com todos • Conhecer seus modelos • Conhecer suas ferramentas • Adaptação local • Comunicação aberta e honesta • Trabalhar com o instinto das pessoas

Descrição	Centrais	Suplementares
Práticas	• Participação ativa daqueles que suportam o projeto • Aplicar os artefatos corretos • Propriedade coletiva • Considerar a testabilidade • Criar vários modelos em paralelo • Criar conteúdo simples • Representar os modelos de forma simples • Apresentar os modelos publicamente • Passar para os outros artefatos • Modelar com os outros membros da equipe • Provar, demonstrar com código • Usar as ferramentas mais simples • Usar um mural para os modelos	• Aplicar normas de modelagem • Aplicar padrões • Descartar modelos temporários • Formalizar os modelos de contrato • Modelar para comunicar • Modelar para entender • Reutilizar recursos existentes • Atualizar somente quando necessário

A MA provê a criação de modelos eficientes, que atendam seu propósito, sejam inteligíveis, suficientemente precisos, consistentes e detalhados. A MA é uma atitude, não um processo prescritivo, sendo um suplemento aos métodos existentes (AMBLER, 2004).

2.2.4 eXtreme Programming (XP)

Segundo (BECK, 2000; CUNNINGHAM, 2002), e*Xtreme Programming* (XP) é uma disciplina de desenvolvimento de software voltada para pequenas e médias equipes, onde os requisitos são vagos e mudam freqüentemente. Desenvolvido por *Kent Beck*, *Ward Cunningham* e *Ron Jeffries*, a XP tem como principal tarefa a codificação, com ênfase menor nos processos formais de desenvolvimento, com uma maior disciplina de codificação e testes. Tem como princípios a comunicação, simplicidade, *feedback* de usuários e coragem dos desenvolvedores.

A XP valoriza a criação e automação dos testes, sendo criados antes, durante e depois da codificação. É flexível a mudanças de requisitos, valorizando o *feedback* com o usuário e a qualidade do código fonte final (ASTELS; MILLER; NOVAK, 2002).

A idéia principal da XP é a criação de software de alta qualidade, abandonando todo tipo de *overhead* de processo que não suporte diretamente esse objetivo. A XP é orientada explicitamente a pessoas e vai contra o senso comum do gerenciamento de que as pessoas são peças intercambiáveis dentro do processo de desenvolvimento.

2.2.4.1 Práticas XP

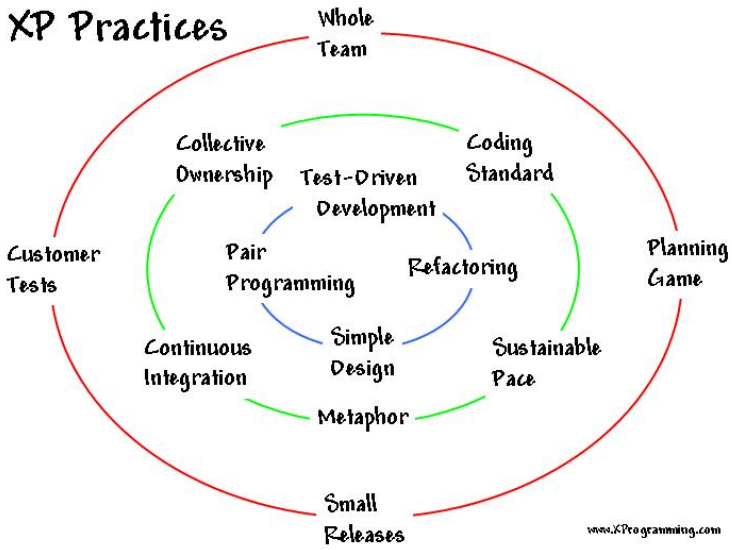

Figura 2.6: Práticas da XP

O funcionamento da XP é baseado em um conjunto de regras e práticas, divididos em quatro grandes grupos: planejamento, desenvolvimento da arquitetura, codificação e testes. As práticas da XP são divididas organizacionais (círculo vermelho), equipe (círculo verde) e pares (círculo azul). São elas (OBJECT MENTOR, 2002; ASTELS; MILLER; NOVAK, 2002):

- Equipe (*Whole Team)*: todos em um projeto XP são partes da equipe, integrando os clientes à equipe de desenvolvimento. É importante salientar que um membro da equipe pode assumir mais de um papel. Na tabela 2.6, tem-se os principais papéis da equipe em XP (ASTELS; MILLER; NOVAK, 2002):

Tabela 2.6: Papéis da XP

Papel	**Participação**
Proprietários do Ouro	É o cliente que paga pelo desenvolvimento do projeto. Equipe Cliente.
Contadores de História	Define os requisitos do software e utiliza o produto final. Equipe Cliente.

Papel	Participação
Aceitantes	São as pessoas que executarão os testes de aceitação. Garantem que cada *release* atenda às necessidades dos contadores de história. Equipe Cliente.
Planejadores	São as pessoas que vêem as necessidades de distribuição. Eles entendem os ciclos de negócios e programam a distribuição. Equipe Cliente.
Chefe	É responsável pela liderança do projeto, supervisionando as equipes. Equipe Cliente.
Gerente	Garante os recursos necessários, sendo o escudo da equipe em relação aos problemas do projeto. Equipe de Desenvolvimento.
Testadores	Ajudam o cliente com os testes de aceitação, executando-os uma vez por dia. Equipe de Desenvolvimento.
Analistas	Ajudam o cliente na definição dos requisitos. Equipe de Desenvolvimento.
Técnico	Orienta a equipe, controlando a aplicação da XP. Equipe de Desenvolvimento.
Analista de Negócio	Coleta as métricas do projeto, controlando o seu andamento. Equipe de Desenvolvimento.
Programadores	Responsáveis pela implementação do código. Equipe de Desenvolvimento.

- Clientes no Local (*Client on Site)*: Para total funcionamento da XP, é necessário que o cliente se integre à equipe de desenvolvimento;
- Jogo do Planejamento (*Planning Game)*: São definidas estimativas e prioridades. Ótimo *feedback* para que cliente possa dirigir o projeto, podendo-se ter uma idéia clara do avanço do projeto minimizando os riscos. Nesta prática é onde são definidas as histórias que descrevem as funcionalidades a serem implementadas;
- Testes de Aceitação (*Customer Tests)*: São testes elaborados pelo cliente, sendo os critérios de aceitação do software. Devem ser rodados a cada interação futura. Oferecem *feedback* do desenvolvimento da aplicação;
- Pequenas entregas (*Small Releases)*: Disponibiliza a cada interação o software 100% funcional. Desta forma, é disponibilizado pequenas versões para que o cliente possa obter o seu ganho o mais cedo possível, minimizando os riscos;
- Posse Coletiva (*Collective Ownership)*: Em um projeto XP, qualquer dupla de programadores pode melhorar o software a qualquer momento. O código tem um único dono: a equipe. Todos compartilham a responsabilidade pelas alterações;

- Integração Contínua (*Continuous Integration)*: XP mantém o projeto integrado continuamente, expondo o estado atual de desenvolvimento (viabiliza lançamentos pequenos e frequentes), oferencendo um *feedback* sobre todo o sistema, em qualquer etapa do processo;
- Metáfora (*Metaphor)*: A equipe mantém uma visão compartilhada do funcionamento do software. Serve de base para estabelecimento dos padrões de codificação;
- Ritmo Saudável (*Sustainable Pace)*: Os projetos XP procuram obter a maior produtividade dos programadores e entregar o software na melhor qualidade possível, obtendo a satisfação do cliente. Para isso há uma grande preocupação com o cronograma. Definir um número de horas de trabalho é fundamental para o maior rendimento da equipe;
- Padrões de Codificação (*Coding Standards)*: Todo o código escrito segue o padrão definido pela equipe, facilitando a comunicação e refinamento do *design*;
- Testes (*Test-Driven Development)*: Todo o desenvolvimento XP é guiado por testes. Os testes dão mais segurança, coragem para mudar. Os testes são a base do *feedback*;
- Programação em Pares (*Pair Programming)*: Todo o desenvolvimento é feito em pares, obtendo uma melhor qualidade do *design*, código e testes, uma revisão constante do código e uma maior comunicação (OBJECT MENTOR, 2004);
- *Design* Simples (*Simple Design)*: O código está, a qualquer momento, na forma mais simples para a realização dos testes. Esta prática está presente em todas as outras etapas da XP.
- Refinamento (*Refactoring)*: O *design* é melhorado continuamente através do refinamento. Em XP, o código é o próprio *design*. O refinamento é um processo formal realizado através de etapas reversíveis, melhorando a estrutura do código sem alterar sua função (FOWLER, 1999);
- *Spikes* de Planejamento *(Spikes Solution)*: Os *spikes s*ão pequenas investigações de tecnologias que podem se agregar ao projeto, envolvendo pesquisas relacionadas a arquitetura, *frameworks* e componentes, algoritmos, Banco de Dados, Servidores *Web*, entre outros.
- Reuniões em Pé *(Stand-Up Meeting)*: São reuniões relativamente curtas, realizadas em pé, abordando tarefas que foram realizadas no dia anterior e tarefas que serão realizadas hoje. Geralmente estas reuniões são realizadas pela manhã. São reuniões breves e sem perda do foco do trabalho, mantendo a equipe em sintonia.

2.2.4.2 Uso da XP

A XP é uma disciplina ágil, que requer total integração da equipe. Por isso, os valores de comunicação, simplicidade, coragem e *feedback* são sempre levados em consideração antes da aplicação da disciplina. Como a XP é voltado para testes, é muito importante a projeção dos testes antes da codificação. Isto permite o *refactoring* sem qualquer temor, dando mais segurança ao engenheiro de software.

Entretanto fica complicado aplicar a XP quando os projetos são desenvolvidos por equipes grandes e espalhadas geograficamente, dificultando a comunicação. Outra situação onde se torna complicado aplicar a XP é quando não se tem controle sobre o código e quando o *feedback* é demorado, sem uma comunicação eficiente (ASTELS; MILLER; NOVAK, 2002).

Outras questões que dificultam o uso da XP são barreiras culturais, como alteração de código de terceiros e relevância de hábitos antigos, procurando manter as coisas simples. A XP também não pode ser aplicado quando o cliente quer uma especificação detalhada do sistema antes de começar o desenvolvimento.

Embora as principais literaturas relacionadas a XP (BECK, 2000; JEFFRIES; ANDERSON; HENDRICKSON, 2001; BECK; FOWLER, 2001) não descrevam os artefatos de forma explícita, é possível identificar cerca de 30 artefatos. Existem inúmeras referências de XP abordando outras práticas, além das referenciadas neste trabalho[12].

A XP tem obtido um crescimento contínuo como disciplina de desenvolvimento dentro da indústria do software, sendo discutida pelos principais engenheiros de software do mercado. Existem vários trabalhos comparando a XP ao RUP, como (MARTIN, 2002; SMITH, 2002), relacionando características em termos de tempo e alocação de recursos, artefatos, disciplinas e atividades.

2.2.5 Processo APOENA de Desenvolvimento de Software (PADS)

O *Processo APOENA de Desenvolvimento de Software - PADS* é um processo padrão de desenvolvimento de software, incremental e iterativo, dirigido por casos de uso, desenvolvido pela APOENA[13], adotado como guia pela equipe de desenvolvimento.

Este processo foi criado após uma análise da equipe de desenvolvimento junto a outros processos, procurando verificar o que poderia ser adaptado para o PADS, onde o principal objetivo era de documentação das etapas e artefatos, focado no modelo de negócio da empresa.

O PADS surgiu para suprir as seguintes necessidades:

- Ser um processo ágil (AMBLER, 2004);
- Focado no modelo de negócios da empresa;
- Abranger todo o ciclo de vida do software;
- Possuir documentação e acompanhamento das etapas;
- Reduzir custos de projeto;

---

[12] Existem outras práticas XP que estão surgindo. O principal objetivo é manter o foco no projeto de forma ágil. Mais informações em http://www.xprogramming.com e http://www.extremeprogramming.org

[13] O autor é Diretor de Tecnologia da APOENA Software Livre, empresa que trabalha com consultoria e desenvolvimento de software livre e *open-source.*

- Aumentar a integração da equipe;
- Automatizar etapas e artefatos;
- Desenvolvimento dirigido por testes;
- Obter *feedback* prévio do cliente.

O PADS utiliza técnicas de desenvolvimento Orientadas a Objetos, usando como base a RUP (*Rational Unified Process*), alguns artefatos definido pelo PMR de *Craig Larman* e práticas da XP, utilizando a UML *(Unified Modeling Language)* como linguagem de modelagem. Outros artefatos foram integrados ao processo, pelo fato de agregarem informação ao projeto.

As etapas são definidas em fluxo de atividades e artefatos. O PADS é amplamente configurável, permitindo a identificação de artefatos, ferramentas e atividades que se enquadram em cada projeto.

A elaboração dos artefatos e execução das atividades também é apoiada por padrões, modelos e guias, adicionados no processo a fim de garantir um entendimento comum entre os membros da equipe. As verificações e validações dos artefatos são realizadas em cada iteração.

2.2.5.1 Etapas do PADS

O processo de desenvolvimento é incremental e iterativo, dirigido por casos de uso, sendo divido nas seguintes etapas: Elicitação de Requisitos, Análise, Projeto, Implementação e Testes. Cada etapa contém uma série de artefatos e atividades relacionadas. Na tabela 2.7, tem-se a relação das atividades que compõem cada etapa.

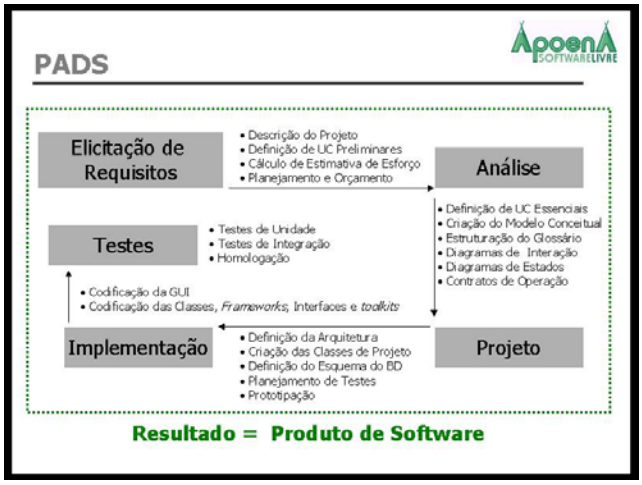

Figura 2.7: Etapas do PADS

Tabela 2.7: Etapas, Atividades e Artefatos do PADS

Etapa	Atividade	Artefatos
Elicitação dos Requisitos	Descrição do Projeto	Para documentação dos requisitos de software, o PADS possui alguns artefatos que norteiam esta atividade: *Descrição do software:* Abrange dados gerais do cliente, descrição textual do software e requisitos de tecnologia e fatores adicionais (segurança, distribuição, concorrência, integração com outros sistemas). *Workflows Relacionados:* Definição de processos de negócio através de *workflows*. *Arquitetura da Aplicação:* Definição da Arquitetura proposta para o software, e que *a posteriori* será validada pelo protótipo. *Relatórios e formulários pré-existentes:* Relatórios e formulários utilizados para automatizar o processo atualmente na empresa. *Histórico de Decisões do Projeto:* Artefato onde ficam registrados todas as decisões do projeto. *Dúvidas do Projeto:* Artefato contendo a documentação dos principais questionamentos relativo ao domínio do problema, que deverão ser feitos ao Cliente. *Layout dos Relatórios:* Artefato contendo uma descrição preliminar dos principais relatórios que o software deverá gerar.
	Definição dos Casos de Uso Preliminares	Registrar os tipos de processos necessários, bem como seus relacionamentos e possíveis transações, através de uma investigação preliminar.
	Cálculo da Estimativa de Desenvolvimento	É estimado o tempo de desenvolvimento necessário para a construção do software, baseado em cálculos de pontos de função por casos de uso.
	Planejamento e Orçamento do Projeto	Após calculado a estimativa de desenvolvimento do software, é realizado o planejamento, identificando potenciais funcionalidades do software, prazos, riscos, *spikes*[14] de planejamento e custos.

[14] *Spikes* de planejamento são investigações sobre tecnologias que podem ser utilizadas no projeto.

Etapa	Atividade	Artefatos
Análise	Definição dos Casos de Uso Essenciais	Utilizando a elicitação de requisitos como base, são identificados os casos de uso essenciais, que são processos que compõem as atividades dos casos de uso preliminares descritos de forma narrativa. Após definidos os casos de uso, é criado o diagrama de casos de uso da aplicação, definindo os atores, processos e seus relacionamentos.
	Criação do Modelo Conceitual	A partir da identificação dos conceitos através dos casos de uso essenciais, é criado o modelo conceitual (modelo de classes com atributos e associações), representando as classes de negócio.
	Estruturação do Glossário	Os termos identificados ao longo do ciclo de vida devem ser documentados no glossário. Isto facilita o entendimento dos termos e amplia a forma de trabalho da equipe.
	Diagramas de Interação	Nesta etapa, são definidos os diagramas de interação (diagrama de seqüência e de colaboração) para definição das operações entre os objetos.
	Diagramas de Estado	É onde são apresentados determinados estados que um objeto assume.
	Contratos de Operação	Define-se também os contratos de operação para os métodos, caso seja necessário.
Projeto	Definição da Arquitetura da Aplicação	É definida a arquitetura da aplicação, com requisitos de tecnologia: padrões de projeto, arquitetura, tipo de SGBD, plataforma, número de usuários, *fine tunning* do BD, requisitos de segurança.
	Criação das Classes de Projeto	São definidas as classes de serviços adicionais e *frameworks* que serão utilizados em conjunto com as classes de negócio.
	Diagrama de Pacotes	As classes de serviços, *frameworks* e classes negócios agrupadas de acordo com sua semântica, em pacotes. É definido um diagrama de visualização das colaborações entre estas classes com identificação de funcionalidades gerais.
	Esquema de SGBD	São identificadas quais classes de negócio serão mapeadas para o SGBD. Após esta definição, é criado o modelo E-R e *scripts* SQL.
	Planejamento de Testes	Através de um estudo sobre alguns artefatos definidos anteriormente (como casos de uso essenciais, diagramas de estado, interação, contratos), são mapeados os testes de unidade e projetados os testes de integração e aceitação do software. É gerado um artefato com o planejamento dos testes a serem realizados.

Etapa	Atividade	Artefatos
Projeto	Prototipação	É criado um protótipo já baseado na arquitetura definida, para posterior codificação de suas funcionalidades. É utilizado um artefato para documentação da verificação do protótipo junto aos clientes.
Implementação	Codificação da GUI *(Graphical User Interface)*	Baseado no protótipo, é que são codificados o controle de navegação e as principais funcionalidades de interface.
	Codificação/ Integração das Classes, Interfaces, *Frameworks* e Componentes	As classes definidas no diagrama de projeto são codificadas segundo padrões de arquitetura e de codificação definidos, juntamente com seus códigos de teste de unidade.
Testes	Testes de Unidade	Nesta atividade são realizados testes localizados na unidade anteriormente planejados, usando ferramentas de teste (Família *xUnit*[15], por exemplo).
	Testes de Integração	São realizados testes funcionais e de dependência entre os módulos, bem como os testes de aceitação do software.
	Homologação	Produto final é homologado após baterias de testes realizados, juntamente com o cliente, gerando uma versão.

Além destas etapas e atividades abordadas, o PADS ainda possui vários modelos de especificação como por exemplo os padrões de codificação. É amplamente configurável, com identificação de atividades, ferramentas e artefatos a serem usados em cada projeto.

Os softwares desenvolvidos com o PADS são projetados em camadas, facilitando assim o desenvolvimento e reaproveitamento do protótipo para a GUI, além de facilitar possíveis atualizações no software. Na figura 2.8, tem-se a definição das camadas dos softwares projetados.

[15] A família *xUnit* é um *framework* de teste de unidade inicialmente criada por *Kent Beck* e *Erich Gamma* para linguagem *SmallTalk*. Atualmente, este *framework* foi portado para várias linguagens. Mais informações em http://www.xprogramming.com/software.htm.

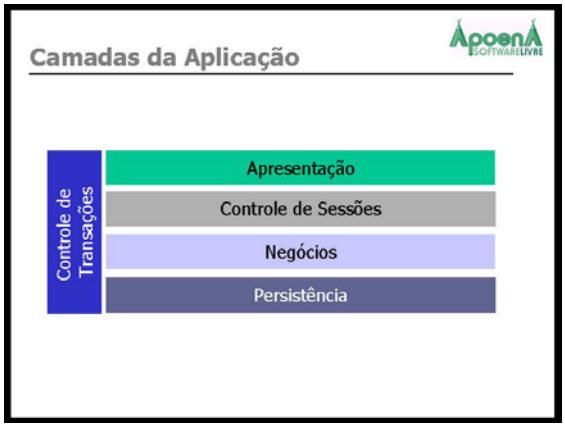

Figura 2.8: Camadas da Aplicação

2.2.5.2 Papéis e Práticas Adotadas

No PADS, são identificados 5 papéis com funcionalidades distintas:

- Analista de Sistemas – Responsável pelo trabalho de investigação dos requisitos e estruturação da solução, trabalhando em todas as etapas do processo.
- Projetista de Software – Responsável pelas decisões de projeto, relacionados à tecnologia, arquitetura, componentes e padrões adotados.
- Engenheiro de Software – Responsável pela validação das decisões de projeto, implementação do protótipo e construção do software.
- Testador – Responsável por planejar os testes juntamente com os demais participantes da equipe (analista, projetista e engenheiro de software) e aplicá-los a cada iteração.
- Gerente – Controla as atividades da equipe e gerencia os recursos para o projeto.

As práticas XP são adotadas quase que integralmente, proporcionando a distribuição e nivelamento da equipe, dependendo apenas das características do projeto. As práticas foram adotadas para melhor produtividade do processo:

- Jogo do Planejamento;
- Teste de Aceitação (definindo com o cliente já no protótipo);
- Pequenos Lançamentos (desde o protótipo até o software de produção);
- Programação em Pares (promoção de sessões de desenvolvimento em pares);
- Integração Contínua;
- Posse Coletiva;
- Desenvolvimento dirigido por Testes (automatização de testes);
- *Refactoring;*

- Padrões de Codificação;
- Metáfora;
- Projeto Simplificado;
- Cliente no Local (em alguns projetos, procura-se substituir está prática através de outras práticas, como *ProxyClient*[16]).

Relacionado a documentação, o foco é utilizar artefatos que realmente sejam importantes para cada projeto.

2.3 Integrando *Frameworks* a Documentação de Software: Trabalhos Relacionados

A integração da abordagem de *frameworks* à documentação de software é muito importante pelo fato de aumentar a produtividade de projeto e código, além de fornecer mecanismos para a geração de artefatos.

Neste caso, serão abordados dois principais aspectos relativos a integração de *frameworks* e documentação de software: primeiramente, a documentação dentro do processo de desenvolvimento de software e, depois, devido as próprias características dos *frameworks*, a geração de artefatos a partir de definições das especificações de alto nível.

A documentação dentro do processo de desenvolvimento de software não faz parte de uma fase definida, mas ocorre durante toda sua existência, em paralelo com outras fases do ciclo de vida. Ela pode ser definida de forma linear ou associativa (BORGES, 1998).

A abordagem realizada de forma linear é seqüencial, onde todo e qualquer documento é desenvolvido para ser lido em uma ordem pré-determinada e inalterável. Desta forma, o engenheiro de software está submetido a um nível de abstração pré-determinado, não tendo a opção de aprofundamento do conteúdo relacionado ao tema. A documentação terá como característica uma disjunção entre os documentos gerados.

O suporte oferecido por uma abordagem associativa, que é baseada em hipertexto, é semelhante a forma linear, porém com o documento contendo referências à documentação de outros artefatos gerados na mesma ou em outra fase do processo de desenvolvimento de software.

Em (BORGES, 1998), tem-se algumas vantagens dos hipertextos em relação a textos lineares:

- Proporcionam conectividade entre as informações;
- Oferecem uma interface compatível com o modo do raciocínio humano;
- Permite ao usuário definir o nível de abstração desejado, navegando entre a documentação dos artefatos.

---

[16] Outra prática conhecida na comunidade XP, onde um membro da equipe faz o papel do cliente, apoiado por reuniões diretas com os clientes, uso de protótipos.

Como as etapas de desenvolvimento de software são inter-relacionadas, os textos lineares apresentam uma deficiência: a falta de capacidade de expressar a fronteira deste relacionamento. Portanto, se torna difícil a utilização de um único texto linear como documentação de todo o processo, não permitindo a visualização individual de cada etapa, dificultando a localização da informação e por outro lado utilizando vários textos lineares com a documentação de cada etapa, não fornecendo uma visão global do processo.

A abordagem de *frameworks* dentro deste contexto enfoca o tratamento de *templates* e definições das características dos artefatos, facilitando e acelerando o processo de documentação.

Relativo à geração de artefatos, os *frameworks* possuem características semelhantes aos geradores de artefatos (FRANCA; STAA, 2001). Um gerador de artefatos é uma ferramenta de software que produz um artefato a partir de uma especificação de alto nível.

Em um processo de desenvolvimento evolutivo, que utiliza a geração de artefatos como recurso, preconiza que a manutenção seja sempre e integralmente realizada na sua especificação e não nos artefatos gerados. Cada ciclo produz uma versão executável do artefato.

As similaridades encontradas entre *frameworks* e geradores de artefatos, sob a ótica de *frameworks* estão relacionadas as partes fixas[17] e variáveis[18] do artefato gerado. A atividade de configuração de *hot-spots* de um *framework* corresponde ao fornecimento da especificação do artefato que vai ser gerado. A instanciação do *framework*, no contexto do gerador, corresponde aos mecanismos de geração. Relacionado a implementação, os *frameworks* utilizam mecanismos OO, e os geradores utilizam outros mecanismos (FILETO; MEIRA; COSTA; MASSHURÁ, 1996).

Dentro dos contextos de *frameworks*, documentação de software e geradores de artefatos, procurou-se selecionar algumas ferramentas com estas características, baseadas na documentação de artefatos UML, geração de artefatos e exportação de formato HTML (*Hypertext Markup Language*), que são apresentadas a seguir.

2.3.1 FrameDoc

O *FrameDoc* é um *framework* de documentação de componentes reutilizáveis, que utiliza tecnologias de multimídia e padrões, desenvolvida por Leonardo Murta (MURTA, 1999).

Modelado com o padrão MVC (*Model-View-Controller*) (GAMMA; HELM; JOHNSON; VLISSIDES, 2000), onde as classes que armazenam as informações (modelos) são separadas das classes que representam as informações (visões) e das classes que manipulam as informações (controlador).

O *FrameDoc* trabalha com o conceito de *templates*. Estes *templates* armazenam os atributos, sendo interpretado como um meta-documento.

---

[17] *frozen-spots*

[18] *hot-spots*

O suporte para documentação no contexto do *FrameDoc* pode ser dividido em quatro etapas (MURTA, 1999):

1. Instanciação do *framework*, relacionada com as atividades necessárias para o registro de componentes que deseja documentar;
2. Definição de *templates* de documentação, onde deve ser executada pelo configurador de ambiente de desenvolvimento, definindo quais informações são necessárias para a documentação de componentes;
3. Criação e atualização da documentação, relacionada com a atividade de empacotamento e adaptação, permitindo a documentação de componentes em vários níveis, de genéricos a específicos;
4. Visualização da documentação, relacionada com a compreensão do componente.

Este *framework* trabalha com a associação de categorias, vinculando os componentes à sua representação semântica[19]. Desta forma, se for necessária a atualização de um determinado modelo implica em atualizar a sua documentação.

O *FrameDoc* possui uma GUI de registro dos componentes, permitindo a geração da documentação em formato HTML, incluindo sons, imagens e vídeos.

Como é uma ferramenta de documentação de componentes associando-os a uma dada categoria, permite a integração com qualquer processo de desenvolvimento.

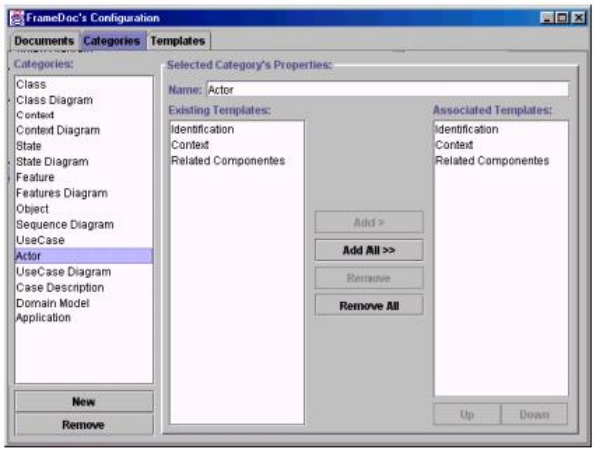

Figura 2.9: FrameDoc

2.3.2 eDoc

O *eDoc* é uma ferramenta de documentação de modelos UML, criada por Sabrina Lobo, Vinícius Teles e Rodrigo Oliveira da ImproveIt[20] (LOBO; OLIVEIRA; TELES, 2001). O *eDoc* foi criado a partir de uma tese de doutorado da FEA/USP e atualmente é

[19] Por exemplo, o termo *advogado* é vinculado a categoria *ator* ou o componente *Aluno* é vinculado a categoria *classe*. Neste caso, as categorias são os casos de uso, atores, classes, atributos, métodos, entre outros.

[20] Mais informações em http://www.improveit.com.br

um produto utilizado para complementar ferramentas de modelagem UML como *Rational Rose*[21] e *Poseidon*[22], por exemplo.

O *eDoc* surgiu para melhorar a qualidade de documentação relacionados aos seguintes problemas (TELES, 2002):

- Má utilização da UML;
- Documentação realizada em editores de texto (*MS-Word)*;
- Diferentes versões do mesmo do documento;
- Inconsistência e dificuldade de navegação entre os elementos.

Utiliza *templates* para casos de uso, levantamento de requisitos, classes, entre outros, que são gerenciados pela ferramenta durante todo o processo, permitindo a integração com outros artefatos. Desta forma, permite uma rastreabilidade dos requisitos através dos casos de uso.

Permite o acesso das informações através de todos os engenheiros envolvidos no projeto, mantendo a documentação centralizada e disponibilizada para equipe através da *Web*, garantindo sua consistência. É uma ferramenta multiusuário, permitindo o trabalho com várias equipes e projetos vinculados. Os *templates* definidos para documentação dos artefatos estão baseados na RUP.

O *eDoc* possui um *plug-in* para importação de modelos gerados pelo *Rational Rose*. Permite a geração de relatórios de consolidação de requisitos, casos de uso e atores (LOBO; OLIVEIRA; TELES, 2001).

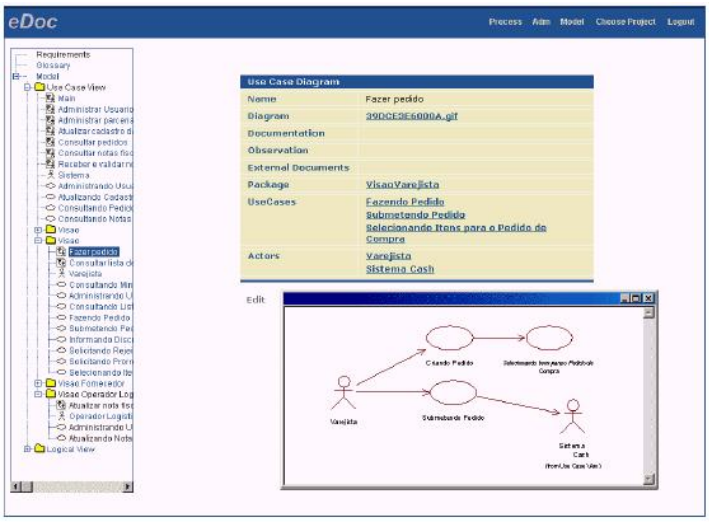

Figura 2.10: eDoc

2.3.3 Discussão

Para comparação entre as ferramentas *eDoc* e *FrameDoc*, foram selecionados alguns critérios e características funcionais que são cobertas totalmente, parcialmente ou não cobertas pelas ferramentas descritas anteriormente. Estes critérios são:

---

[21] Mais informações em http://www.rational.com

[22] Mais informações em http://www.gentleware.com

- *Exportação da documentação em formato portável:* indica se a ferramenta permite salvar os documentos gerados em um formato aberto;
- *Utilização de hiperlinks na documentação:* indica se a ferramenta permite a criação de referências dos artefatos gerados ou partes de componentes da documentação para outros componentes na *Web*;
- *Criação e configuração de padrões de documentação:* indica se a ferramenta permite a criação de *templates* ou formato pré-definido e configurável, formando uma documentação padronizada;
- *Geração de documentação modular:* indica se a ferramenta permite a geração de documentos separados para artefatos distintos, evitando o acúmulo de informações em documentos monolíticos;
- *Geração da documentação implícita ao artefato:* indica se a ferramenta permite a exibição de características inerentes ao próprio artefato na sua documentação;
- *Tecnologias:* Indica as tecnologias que foram utilizadas para desenvolver a ferramenta;
- *Geração de Artefatos:* Compreende a geração de artefatos, além de documentação.
- *Integração com outras ferramentas:* indica se a ferramenta possui algum mecanismo de integração com outras ferramentas.
- *Dependência de Processos:* indica o grau de dependência da ferramenta com processos de desenvolvimento.

A seguir, tem a tabela 2.8 resume uma comparação das ferramentas *eDoc* e *FrameDoc* segundo critérios acima apresentados.

Tabela 2.8: Dicussão entre as Ferramentas de Documentação

Critérios	***FrameDoc***	***EDoc***
Exportação da documentação em formato portável	Exporta a documentação no formato HTML	Exporta a documentação no formato HTML
Utilização de *hiperlinks* na documentação	Permite	Permite
Criação e configuração de padrões de documentação	Permite a criação de *templates* associados a categorias	Possui *templates* internos para documentação dos artefatos
Geração de documentação modular	Permite a geração de documentação de qualquer artefato de forma isolada	Ferramenta de integração, não permitindo a geração parcial da documentação
Geração de documentação implícita ao artefato	Permite	Permite

Critérios	*FrameDoc*	*EDoc*
Tecnologias	*Java Swing*, *com* serialização, JDBC/ODBC e conexão com o GOA++[23]	Componentes COM (desenvolvidos em *Delphi*), ASP e SQL *Server*
Geração de Artefatos	Somente a documentação	Relatórios de requisitos, casos de uso e atores
Integração com outras ferramentas	Não	Integração com a *Rational Rose*
Dependência de Processos	Independente de processos	RUP

[23] Mais informações em http://www.cos.ufrj.br/~goa/

3 FRAMEWORKDOC: CARACTERÍSTICAS E FUNCIONAMENTO

Neste capítulo são apresentadas as principais características do *FrameworkDoc*.

O *FrameworkDoc* é uma ferramenta projetada para realizar a documentação e geração de artefatos de software utilizados no PADS. Como o PADS é baseado em processos já utilizados na indústria, o *FrameworkDoc* passa a atender também estes processos, bem como outros que utilizem UML como linguagem de modelagem. A seguir, são abordados aspectos como estrutura, componentes, artefatos, tecnologia e arquitetura da ferramenta.

3.1 Estrutura

O *FrameworkDoc* foi desenvolvido utilizando uma abordagem associativa de documentação (BORGES, 1998), baseada em *hipertexto.*

Na figura 3.1 tem-se a estrutura do *FrameworkDoc*, com seus principais componentes.

Figura 3.1: Estrutura do *FrameworkDoc*

São abordados dois aspectos de documentação e geração de artefatos pelo *FrameworkDoc*: aplicação e documentação técnica.

A figura 3.2 apresenta o funcionamento do *FrameworkDoc*, com o apoio da ferramenta *Apache ANT*, utilizada para estruturar o projeto de documentação, criando diretórios e gerando documentação de fonte, vindo a ser publicado pelo *FrameworkDoc.*

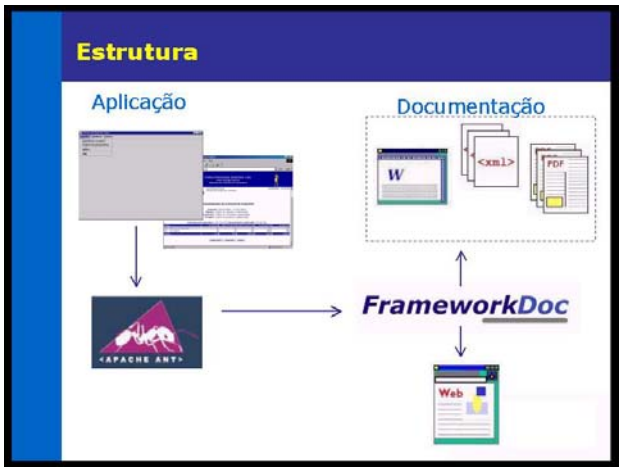

Figura 3.2: Funcionamento do *FrameworkDoc*, com *Apache ANT*

Na figura 3.3 tem-se um pequeno trecho do arquivo *fdoc-ant.xml*, arquivo de *build* do *Apache ANT*, utilizado para auxiliar a configuração da documentação de artefatos do projeto.

No contexto de aplicação, a documentação realizada é referente a ajuda sensível ao contexto do software a ser construído, dicionários de internacionalização I18N[24] (*Internacionalization*) e arquivo de *build*[25] do projeto.

No contexto de documentação técnica, abrange desde documentação de processos, aproveitados pela ajuda sensível ao contexto, até artefatos UML, *frameworks*, componentes utilizados e arquiteturas da aplicação.

[24] Mais informações em http://www.i18n.com/

[25] Arquivo com as principais tarefas de projeto, automatizadas pela ferramenta *Apache* ANT. Estas tarefas podem ser de criação de diretórios, compilação de fontes, integração, execução de testes automatizados, FTP, envio de e-mails, geração de relatórios, entre outros.

```
<?xml version="1.0" ?>
<project name="FrameworkDoc build file" default="main" basedir=".">
        <!-- ############################################ -->
        <!-- Dados de Identificacao                       -->
        <!-- ############################################ -->
        <property name="Name" value="Guilherme Lacerda"/>
        <property name="Email" value="guilherme@apoenasoftwarelivre.com.br"/>
        <property name="Company" value="APOENA Software Livre"/>
        <property name="Site" value="http://www.apoenasoftwarelivre.com.br"/>
        <!-- ############################################ -->
        <!-- Dados do projeto a ser gerado                -->
        <!-- ############################################ -->
        <property name="DefaultProject" value="Contratos"/>
        <property name="project.dir" value="contratos"/>
        <property name="artifact.dir" value="artifacts"/>
        <property name="images.dir" value="images"/>
        <property name="xml.dir" value="xml"/>
        <property name="xsl.dir" value="xsl"/>
        <!-- ############################################ -->
        <!-- Tarefas                                      -->
        <!-- ############################################ -->
        <target name="makedir">
                <mkdir dir="${project.dir}"/>
                <mkdir dir="${project.dir}/${artifact.dir}"/>
                <mkdir dir="${project.dir}/${images.dir}"/>
```

Figura 3.3: Trecho do arquivo *fdoc-ant.xml*

Os diretórios estão organizados da seguinte forma:

- *Configuração:* diretórios de configuração da ferramenta, contendo arquivos *.properties*, CSS, JSP, imagens, arquivos de *log*, entre outros;
- *Projetos:* diretórios específicos dos projetos, gerados automaticamente pelo *Apache ANT*, através do arquivo *fdoc-ant.xml*;
- *Classes e Libs:* classes e bibliotecas *Java*, utilizadas pela ferramenta.

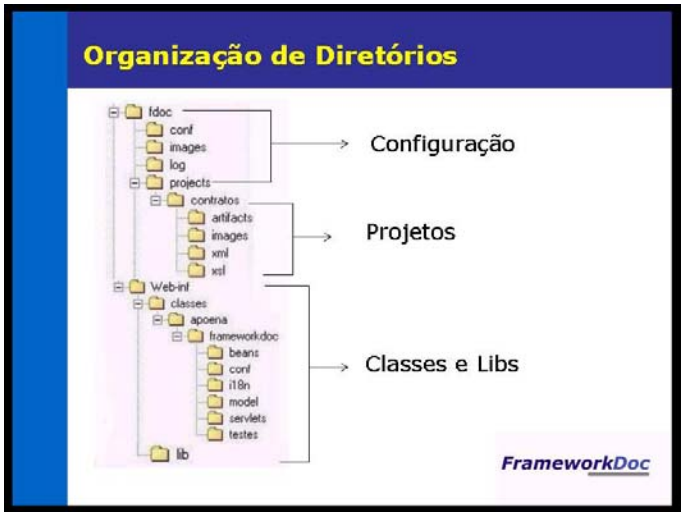

Figura 3.4: Organização de Diretórios

Os principais artefatos encontrados e gerados no *FrameworkDoc* são:

- I18N para *Java*, PHP[26] e C/C++;
- Integração com documentação de código, gerada automaticamente[27] por ferramentas de documentação como *JavaDoc* e *PHPDoc*;
- Ajuda on-line da aplicação, aproveitada para definição das funcionalidades do software;
- Apresentação do software, com logotipo, objetivos e estrutura, baseado no artefato de descrição de projeto;
- Definição da arquitetura da aplicação e *workflows* relacionados;
- Descrição de histórico de decisões de projeto;
- Publicação de testes de aceitação, definidos pelo cliente;
- *Layout* de relatórios da aplicação a ser desenvolvida;
- Homologação do software junto ao cliente e controle de suporte realizado;
- Descrição de *frameworks* e componentes utilizados no projeto;
- Definição do Banco de Dados, através de modelo E-R e descrição do Banco de Dados;
- Definição de padrões de codificação utilizados pela equipe;
- Integração de casos de uso com diagramas, tanto em nível preliminar como essencial, utilizados também na geração automática do plano de testes;
- Descrição de outros artefatos UML, com textos explicativos relacionados;
- Geração de arquivo de *build* de projeto, para a ferramenta *Apache ANT*.

3.2 Tecnologias

Para desenvolvimento do *FrameworkDoc* foram selecionadas as tecnologias *Java* e XML, devido a facilidade de utilização e integração destas, aliado aos principais requisitos da aplicação, que são geração de artefatos e documentação em formato *hipertexto*.

3.2.1 Apache TomCat

O Apache *TomCat* é um servidor *Web* que possui uma implementação oficial para as especificações *Java Servlets* e *JavaServer Pages (JSP)* (Apache Jakarta, 2004). As

[26] Mais informações em http://www.php.net

[27] A documentação do código é gerada automaticamente através de ferramentas de documentação de código, como *JavaDoc* e *PHPDoc*.

especificações de *Servlets* e JSP são desenvolvidos e implementados pela SUN[28] e auxiliado pela *Java Community Process*[29] (JCP).

O *TomCat* é desenvolvido como um projeto aberto e participativo, controlado pelo *Apache Consortium*.

3.2.2 Java

A tecnologia *Java* (SUN MICROSYSTEMS, 2004) é uma plataforma de desenvolvimento de software, que inclui uma máquina virtual (*Java Virtual Machine - JVM*) e a linguagem de programação (ECKEL, 1997; DEITEL; DEITEL, 2000). Atualmente, *Java* possui três plataformas de desenvolvimento, quais sejam (AHMED; UMRYSH, 2002):

- *Java 2 Platform, Micro Edition* (J2ME), utilizada para desenvolvimento de software em dispositivos incorporados, como telefones, palm tops, entre outros;
- *Java 2 Platform, Standard Edition* (J2SE), também conhecida como *Java Development Kit* (JDK), que inclui APIs[30] de acesso a BD, programação em redes, programação distribuída, GUI, *streams*, entre outras;
- *Java 2 Platform, Enterprise Edition* (J2EE), utilizada para desenvolvimento de software corporativo e distribuído, com APIs para *Web services*, componentes, entre outros. É designado para utilização em conjunto com o J2SE.

Para desenvolvimento do *FrameworkDoc* foi utilizada a plataforma *Java* J2SE e J2EE, com as tecnologias de JSP, *Servlets* e *JavaBeans* (FIELDS; KOLB 2000, JÚNIOR, 2002).

3.2.3 API de Logging (java.util.logging)

A *Java Logging API* fornece um conjunto de classes e interfaces para facilitar o tratamento de *log* nas aplicações, fornecendo gerência e suporte a tratamento de erros e exceções dos softwares (Java Logging API, 2004).

O tratamento de *log* se torna cada vez mais essencial no desenvolvimento de software, justamente por auxiliar o rastreamento de erros e alertas que devem ser tratados tanto a nível de usuários finais como desenvolvedores.

3.2.4 JUnit

O *JUnit* é um *framework* de testes de regressão para uso com a linguagem *Java* (JUnit, 2004). Ele foi desenvolvido por *Erich Gamma* e *Kent Beck* a partir do *Kent Beck Testing Framework*, escrito em *Smalltalk* e utilizado no primeiro projeto XP, o C3 (BECK, 2000).

---

[28] Mais informações em http://java.sun.com

[29] Mais informações em http://www.jcp.org

[30] *Application Programming Interface*

Um teste de unidade é uma coleção de testes projetados para verificar o comportamento de uma unidade simples. Em OO, esta unidade é a classe (HUNT; THOMAS, 2003).

3.2.5 Apache ANT

O *Apache ANT* é uma ferramenta de *building* (compilação) de software (Apache ANT, 2004; HATCHER; LOUGHRAN, 2003), com o funcionamento similar a ferramenta *make* encontrada em sistemas operacionais *Unix*, mas com funcionalidades adicionais e extensíveis.

Independente de sistema operacional, o *Apache ANT* não faz chamadas ao sistema operacional, utilizando uma especificação em XML que descreve alvos e tarefas a serem realizadas. Estas tarefas podem ser de criação de diretórios, compilação de fontes, integração, execução de testes automatizados, FTP, envio de e-mails, geração de relatórios, entre outros.

3.2.6 XML

A tecnologia XML (*eXtensible Markup Language*) é uma metalinguagem de marcação definida pela W3C[31], contendo regras definidas pelo usuário, utilizada para descrição de informações, possibilitando a integração de sistemas (DÉCIO, 2000; SILVA, 2001). Esta capacidade é extremamente importante para armazenamento, recuperação e transmissão de informações. Além da definição dos dados, a tecnologia XML permite definir uma estrutura juntamente com os dados, em um formato textual, acessível por qualquer editor de texto.

As principais características do XML são a *interoperabilidade*, permitindo a comunicação entre os sistemas; *organização* e *personalização*, onde os dados são apresentados de forma hierárquica e organizada separando o conteúdo da apresentação e a *autodescrição dos dados*, onde os *tags* descrevem o conteúdo de uma informação.

Existem inúmeras tecnologias associadas à XML como *DTD*, *XML Schema*, *XPath*, *Xquery*, *XSLT* entre outras (DÉCIO, 2000).

Esta tecnologia é utilizada tanto para armazenamento das informações processadas pelo *FrameworkDoc* como para formatação da apresentação das informações através de XSLT (*eXtensible Stylesheet Language Transformations*) (BROGDEN; MINNICK, 2002; VELOSO, 2003) e possível integração com outras ferramentas de desenvolvimento.

3.3 Projeto

O *FrameworkDoc* é projetado para documentar e gerar artefatos de software, definidos, primeiramente, no PADS. Seu funcionamento é totalmente através da *Web*, utilizando uma arquitetura MVC, conforme figura 3.5.

---

[31] Mais informações em http://www.w3c.org

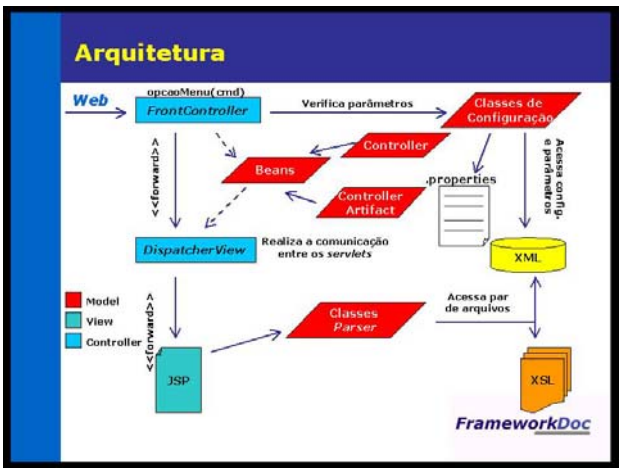

Figura 3.5: Arquitetura MVC

Foi utilizado em seu desenvolvimento alguns padrões de projeto J2EE (ALUR; CRUPI; MALKS, 2002) que facilitam a delegação de responsabilidades para os componentes na camada de apresentação. Os padrões J2EE da camada de apresentação utilizados foram o *Front Controller*, *Service to Worker*, *Dispatcher View* e *Composite View*.

Na camada de *controller*, existem dois *servlets* com funções distintas: o *FrontController* recebe a requisição, faz acesso as instâncias de objetos propriedades, *logs* e menus, definindo o que deve ser exibido. O *servlet DispatcherView* é responsável por chamar os módulos JSP (camada *view*) para exibição dos resultados.

Esta comunicação entre os *servlets* se dá através de *JavaBeans*. No caso do *FrameworkDoc*, existe um *JavaBean* para exibição da lista de artefatos de cada opção do menu principal do processo (*JavaBean Controller*) e outro responsável pela geração e publicação de artefatos (*JavaBean ControllerArtifact*). Para acesso as fontes de dados XML e arquivos *.properties*[32] são usadas algumas classes que estão na camada *model*.

Nos casos de uso estão descritos os requisitos de documentação e geração de artefatos implementados pelo *FrameworkDoc*.(figura 3.6).

[32] Arquivos que contém informações de configuração do *FrameworkDoc*

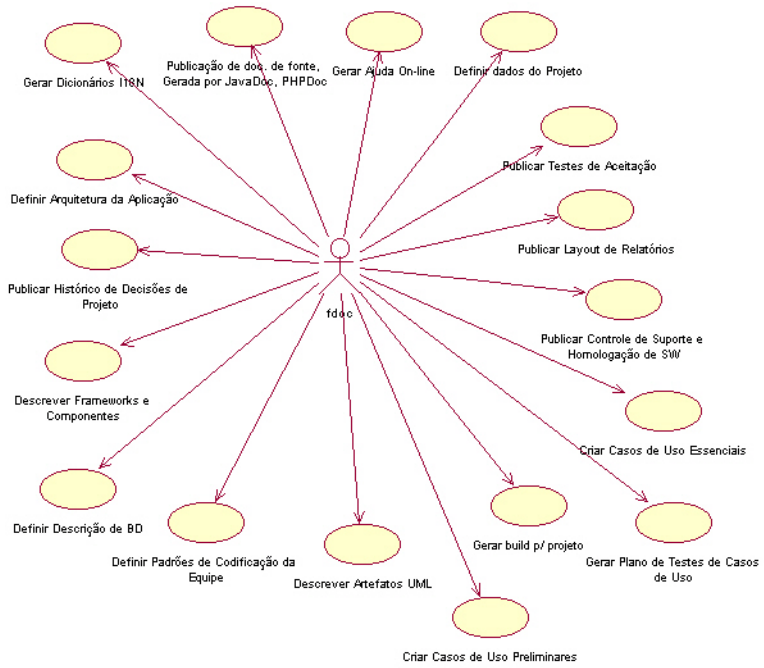

Figura 3.6: Diagrama de Casos de Uso do *FrameworkDoc*

O diagrama de pacotes, descrito na figura 3.7, é utilizado para organizar as classes devido ao seu contexto e seus relacionamentos na aplicação.

No pacote *conf* (figura 3.8) estão as classes de configuração e controle de *log* da ferramenta, que são instanciadas a partir do *servlet FrontController*, gerando um *singleton*[33] de acesso. A classe *Conf* é responsável por ler as propriedades do arquivo *fdoc-conf.properties* e setá-las durante o uso da ferramenta. A classe *Log* implementa a API *java.util.logging*, gerando alertas e erros quando necessário.

[33] Padrão de projeto GoF (GAMMA; HELM; JOHNSON; VLISSIDES, 2000).

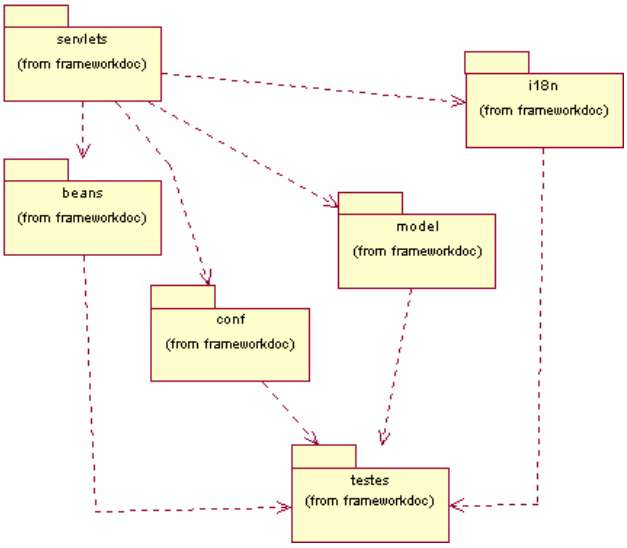

Figura 3.7: Diagrama de Pacotes

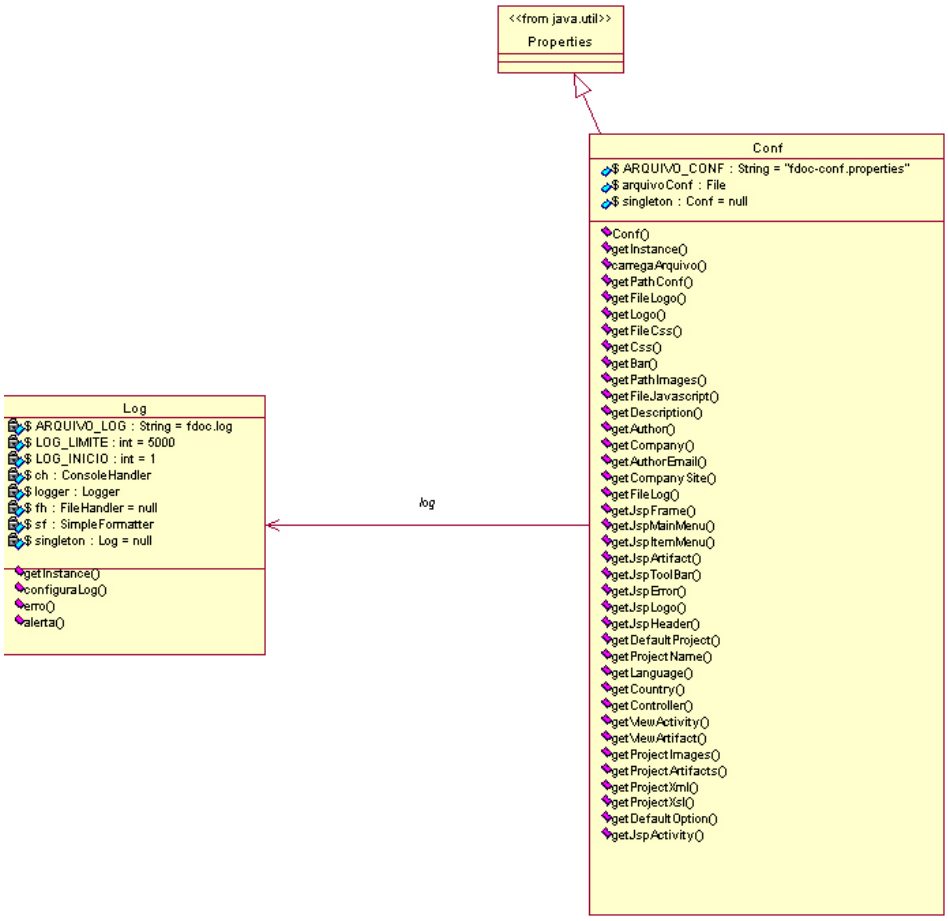

Figura 3.8: Diagrama de Classes – pacote *conf*

No pacote *beans* (figura 3.9), estão as classes de *JavaBean* que são utilizadas para comunicação entre os *servlets*, fazendo uma ligação do componente a ser executado e suas propriedades.

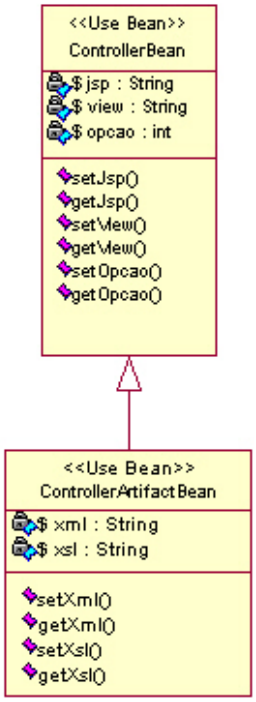

Figura 3.9: Diagrama de Classes – pacote *beans*

No pacote *model* (figura 3.10) estão as classes *MenuVO*, *MenuDAO*, *ItemMenuVO*, *ItemMenuDAO* e *ProcessadorXML*, responsáveis pela configuração dos menus, sub-menus e opções a serem apresentadas. São implementados os padrões de projeto J2EE *Value Object (VO)* e *Data Access Object (DAO)*, com o objetivo de encapsular as opções em suas classes de negócio *(VO)*, gerando listas destas instâncias *(DAO)* armazenadas para utilização da ferramenta. Estas configurações de menus e sub-menus estão no arquivo *fdoc-menu.xml*.

No pacote *servlets* (figura 3.11) têm-se os *servlets FrontController* e *DispatcherView*, anteriormente descritos.

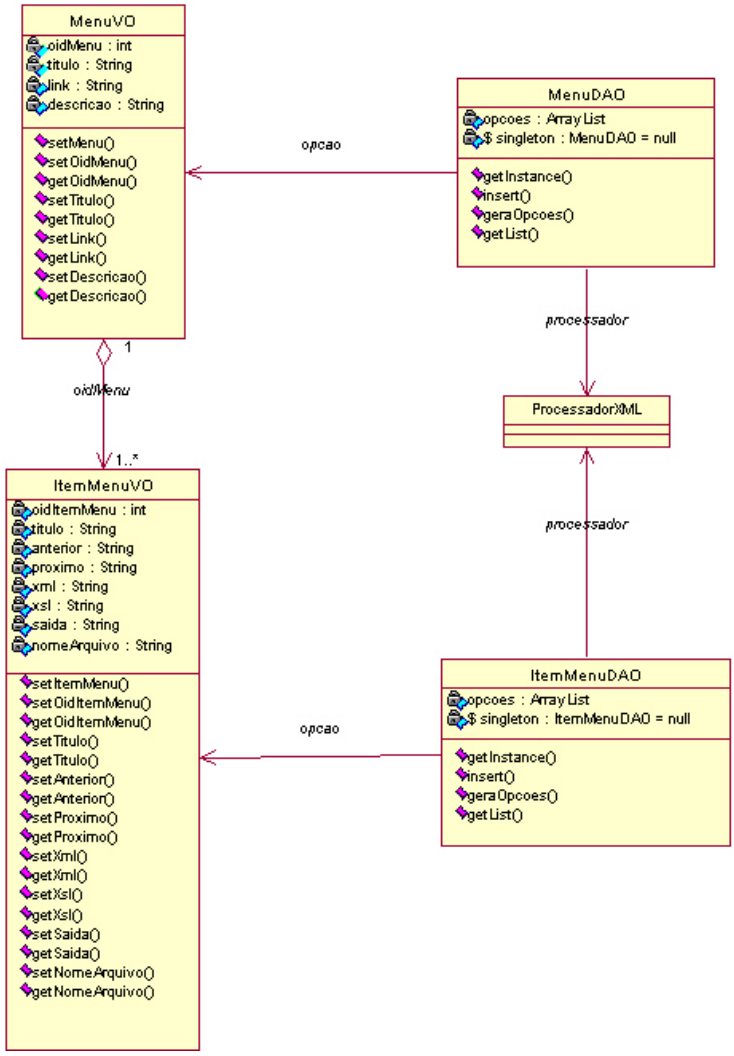

Figura 3.10: Diagrama de Classes – pacote *model*

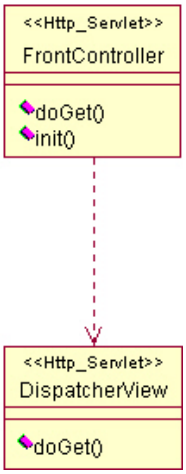

Figura 3.11: Diagrama de Classes – pacote *servlets*

No pacote *i18n* (figura 3.12) estão relacionadas as classes de tratamento do dicionário de internacionalização.

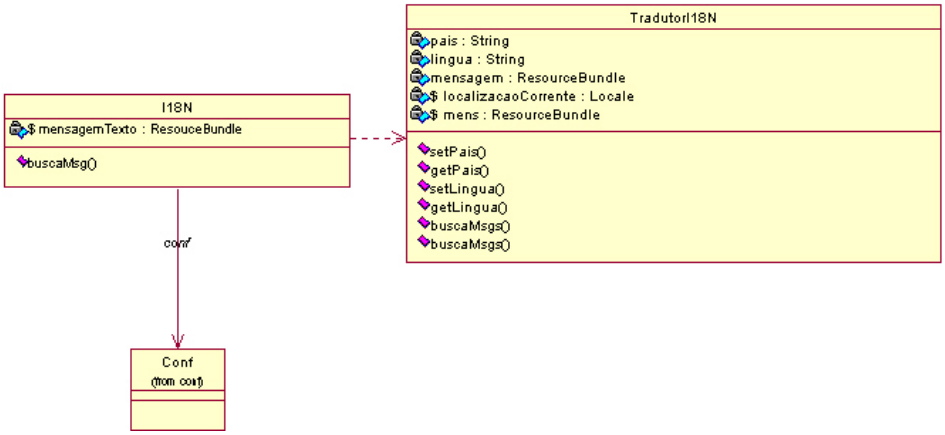

Figura 3.12: Diagrama de Classes – pacote *i18n*

No pacote *testes* (figura 3.13) estão relacionados todos os testes de unidade das classes de negócio e alguns testes de aceitação. É importante citar a utilização da ferramenta *JUnit*, com o desenvolvimento de uma *suite* de testes das classes, mantendo o projeto integrado e testado (figura 3.14).

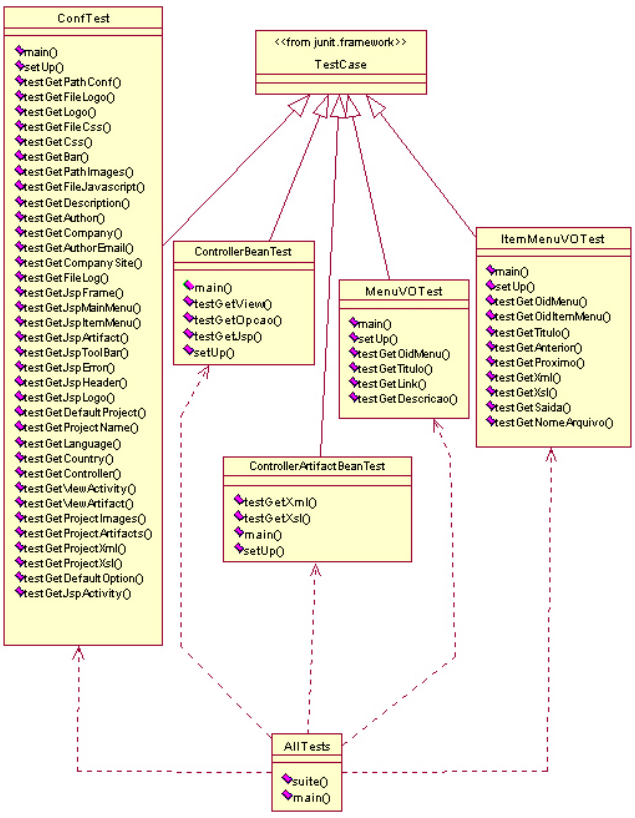

Figura 3.13: Diagrama de Classes – pacote *testes*

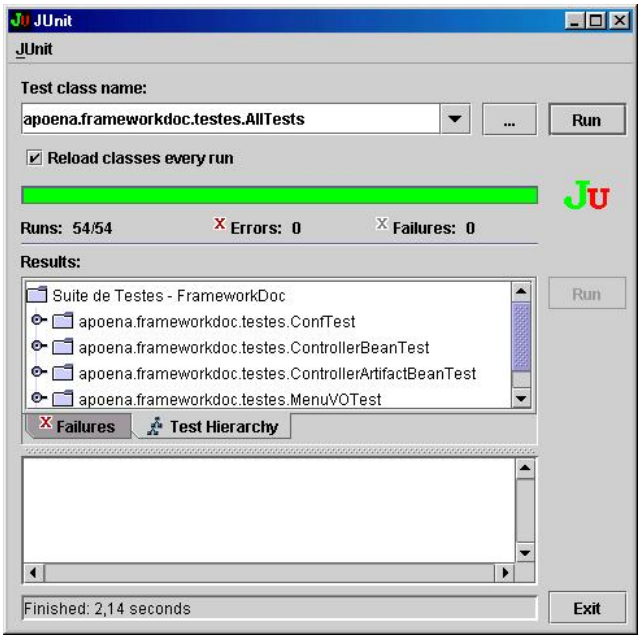

Figura 3.14: *Suite* de testes do *FrameworkDoc*

A GUI do *FrameworkDoc* está dividida em *frames*, cada um com suas funcionalidades, conforme figura 3.15.

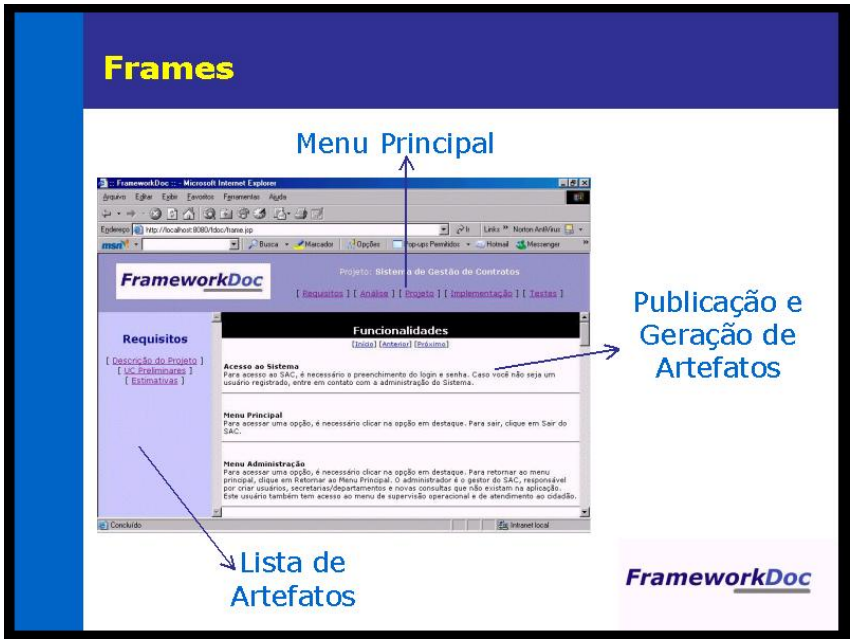

Figura 3.15: *Frames* do *FrameworkDoc*

O *frame* superior é onde são apresentados as etapas do processo a ser documentado, definindo as opções do menu principal. Cada vez que se selecionar uma opção neste *frame* é atualizado o *frame* lateral, listando os artefatos associados a etapa do processo. Quando o artefato é selecionado, o *FrameworkDoc* publica seu conteúdo ou gera outros artefatos a partir do mesmo, exibindo-os no *frame* central.

São invocados vários módulos JSP que trabalham juntos mas com funcionalidades específicas. O diagrama de componentes (figura 3.16), apresenta o relacionamento entre os módulo JSP e o padrão de projeto J2EE *Composite View*. O módulo *frame.jsp* é responsável por invocar todos os outros módulos, fazendo chamadas em *header.jsp* e *activity.jsp*.

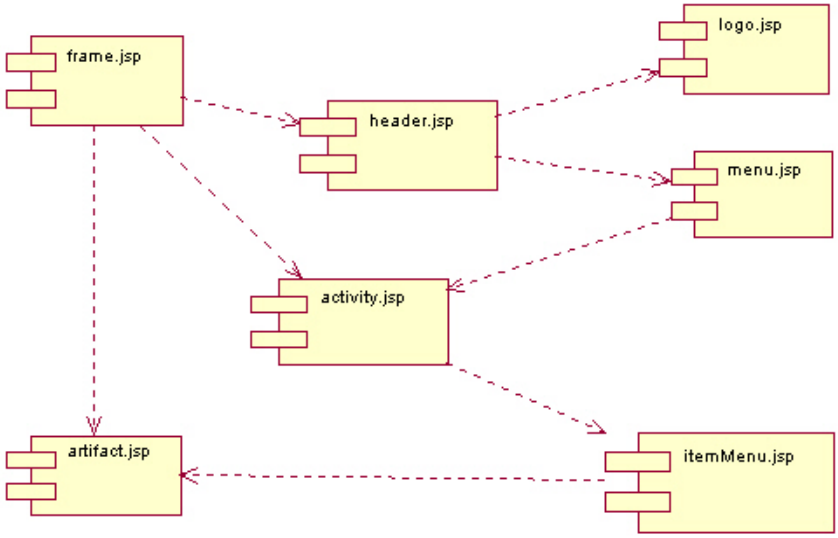

Figura 3.16: Diagrama de Componentes

Na figura 3.17 tem-se a comunicação entre os *servlets FrontController* e *DispatcherView*, utilizando o *bean ControllerBean*, verificando qual lista de artefatos deverá ser exibida ao usuário, dependendo da etapa selecionada no menu principal.

A figura 3.18 apresenta a comunicação entre os *servlets FrontController* e *DispatcherView*, utilizando o *bean ControllerArtifactBean*, para exibir/gerar o artefato selecionado. O *servlet FrontController* é que define, dependendo da ação do usuário, qual *bean* ele deve acionar para comunicar com o *DispatcherView*, que invoca o módulo JSP correspondente.

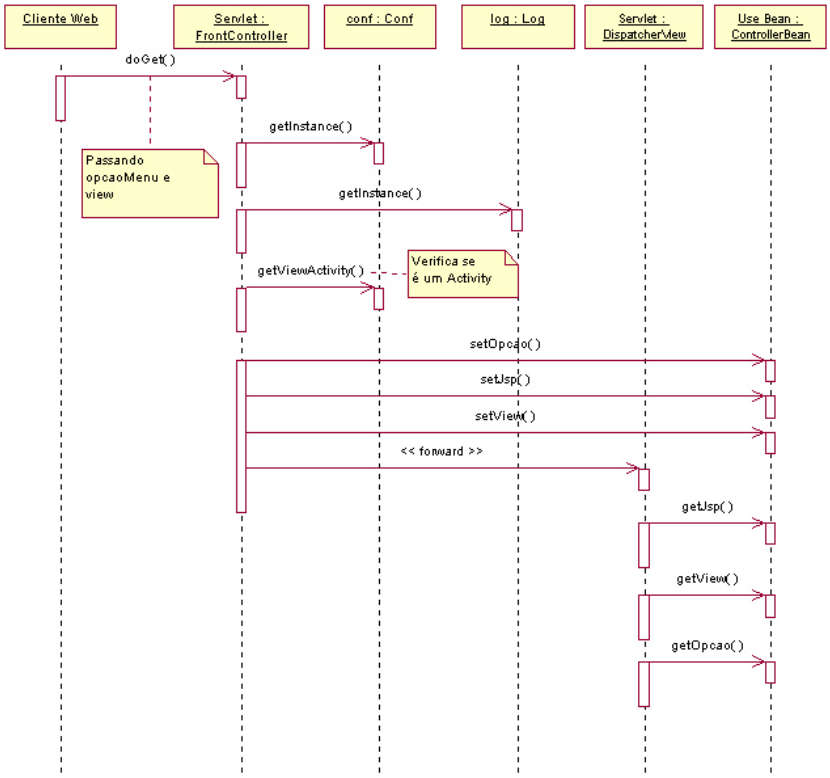

Figura 3.17: Diagrama de Seqüência – Lista de Artefatos

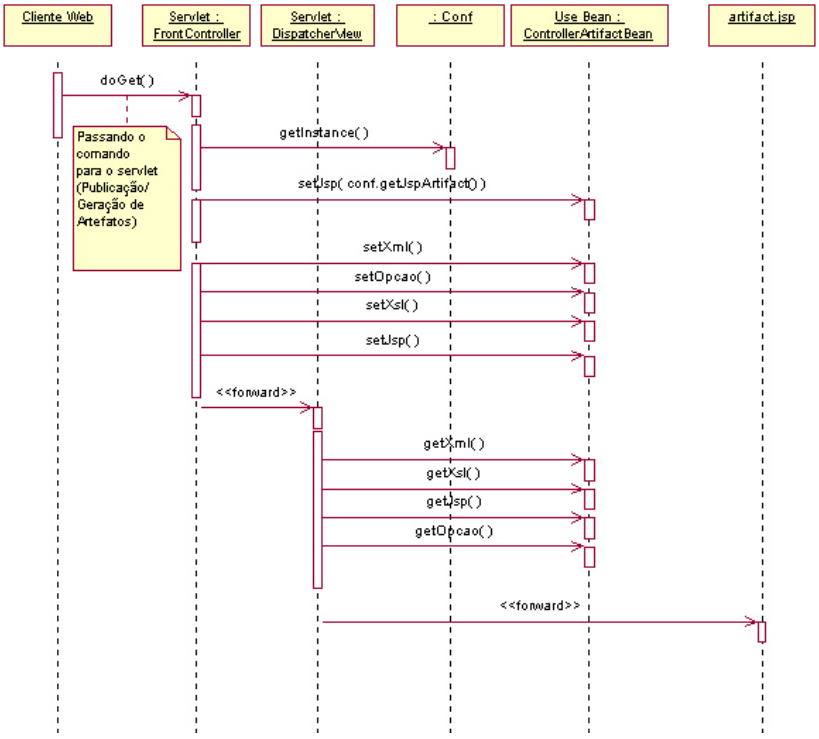

Figura 3.18: Diagrama de Seqüência – Geração/Exibição dos Artefatos

3.4 Usando o *FrameworkDoc:* passo a passo

O *FrameworkDoc* é uma ferramenta de fácil configuração e uso. Para utilizar o *FrameworkDoc* durante um projeto de desenvolvimento, é necessário seguir os seguintes passos:

1. Editar o arquivo *fdoc-ant.xml* (figura 3.3) para criação da árvore de diretórios necessários para o projeto. É importante também relacionar neste arquivo, caso seja necessário, a configuração da documentação de código-fonte;
2. Configurar o arquivo *fdoc-conf.properties*, definindo o projeto atual, no qual a ferramenta deve ser instanciada. A figura 3.19 apresenta um trecho do arquivo *fdoc-conf.properties.*

```
# Arquivo de configuracao do FrameworkDoc
# Por Guilherme Lacerda
# 11/11/2004

# Informacoes sobre o FDOC
fdoc.description = FrameworkDoc
fdoc.author = Guilherme Lacerda
fdoc.company = APOENA Software Livre
fdoc.author.email = guilherme@apoenasoftwarelivre.com.br
fdoc.company.site = http://www.apoenasoftwarelivre.com.br
fdoc.language = pt
fdoc.country = BR

# Outros paths e arquivos utilizados
file.logo = fdoc.jpg
file.css = fdoc.css
path.images = images/
file.javascript = functions.js
path.conf = conf/
file.bar = barra.gif

#configuracoes da ferramenta
view.activity = ACTIVITY
view.artifact = ARTIFACT

# Arquivo de LOG
file.log = fdoc.log

# Arquivos JSP e servlets
file.jsp.frame = /frame.jsp
file.jsp.mainmenu = /menu.jsp
file.jsp.artifact = /artifact.jsp
file.jsp.activity = /activity.jsp
file.jsp.itemmenu = /itemMenu.jsp
file.jsp.toolbar  = /toolbar.jsp
file.jsp.error = /error.jsp
file.jsp.logo =  /logo.jsp
file.jsp.header = /header.jsp
file.servlet.controller = /servlet/FrontController

# Path do projeto atual
default.project = projects/contratos/
project.images = projects/contratos/images/
project.artifacts = projects/contratos/artifacts/
project.xml = projects/contratos/xml/
project.xsl = projects/contratos/xsl/
project.name = Sistema de Gestão de Contratos
default.option = 1
...
```

Figura 3.19: Trecho do arquivo *fdoc-conf.properties*

3. Configurar o arquivo *fdoc-menu.xml* que está dentro do diretório do projeto que está sendo configurado. Este arquivo é onde são definidos as etapas e a lista de artefatos de cada etapa. Criado automaticamente pelo *fdoc-ant.xml*, o arquivo *fdoc-menu.xml* deve ser alterado, ajustando as opções específicas do projeto. Na figura 3.20, tem-se um trecho do *fdoc-menu.xml*. O arquivo *fdoc-menu.xml* é o mais importante da configuração do projeto, pois as informações de acesso a documentação e geração de artefatos estão descritas nele.

```
<?xml version="1.0" ?>
<fdoc>

      <menus>

            <menu id="1">
                  <title>Requisitos</title>
                  <link/>
                  <description>Elicitação de Requisitos</description>
            </menu>

            <!–Continua... →

            <itemmenu id="12" menu="1">
                  <title>UC Preliminares</title>
                  <artifact xml="uc_preliminar.xml" xsl="uc_preliminar.xsl"/>
                  <nav previous="0" next="2"/>
                  <output file=""/>
            </itemmenu>

            <!–Continua... →

      </menus>

</fdoc>
```

Figura 3.20: Trecho do arquivo *fdoc-menu.xml*

O próximo passo é apenas carregar a ferramenta, que trará as configurações definidas nos seus arquivos de propriedades.

3.5 Interface Gráfica do *FrameworkDoc*

A interface gráfica do *FrameworkDoc* foi concebida usando o padrão J2EE *Composite View*, objetivando acessar componentes de forma rápida, facilitando a visualização de artefatos via *Web*. A seguir, são apresentadas algumas telas do *FrameworkDoc*, com a visualização e geração de artefatos.

Na figura 3.21, é apresentado o menu *Requisitos*, com a visualização do artefato *Estimativas de Releases* a serem lançados.

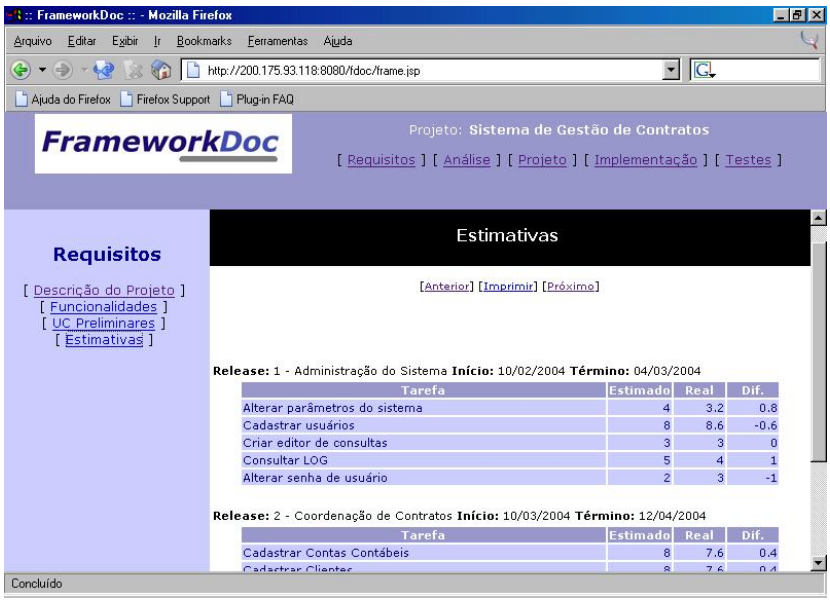

Figura 3.21: Estimativa de *Releases*

O menu *Análise*, apresentado na figura 3.22, permite a visualização do artefato *Casos de Uso Essenciais*, com o seu respectivo diagrama.

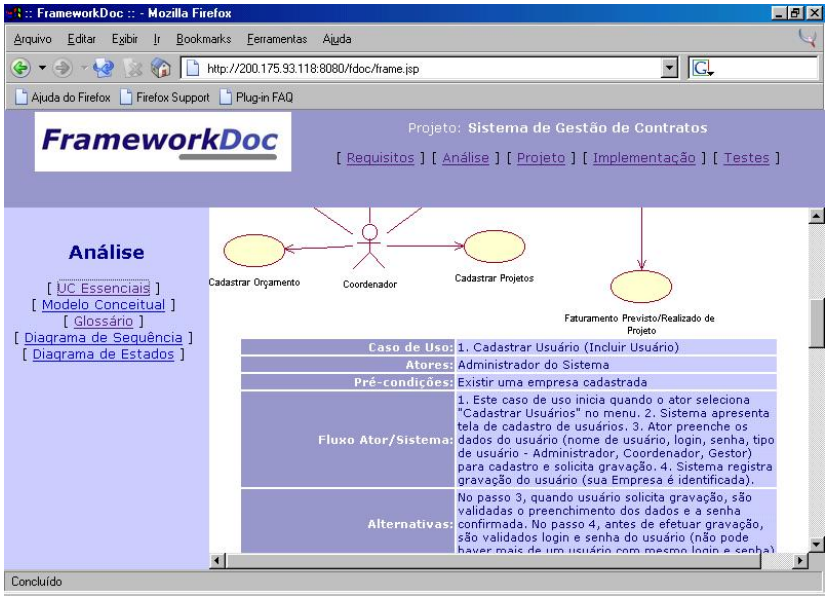

Figura 3.22: Casos de Uso Essenciais

A seguir, é apresentado o menu *Projeto*, com a visualização do artefato *Esquema de Banco de Dados* com o respectivo modelo E-R e descrição das tabelas (figura 3.23). Outro artefato apresentado é o artefato *Histórico de Decisões de Projeto*, que relata as alternativas e decisões tomadas ao longo do projeto (figura 3.24).

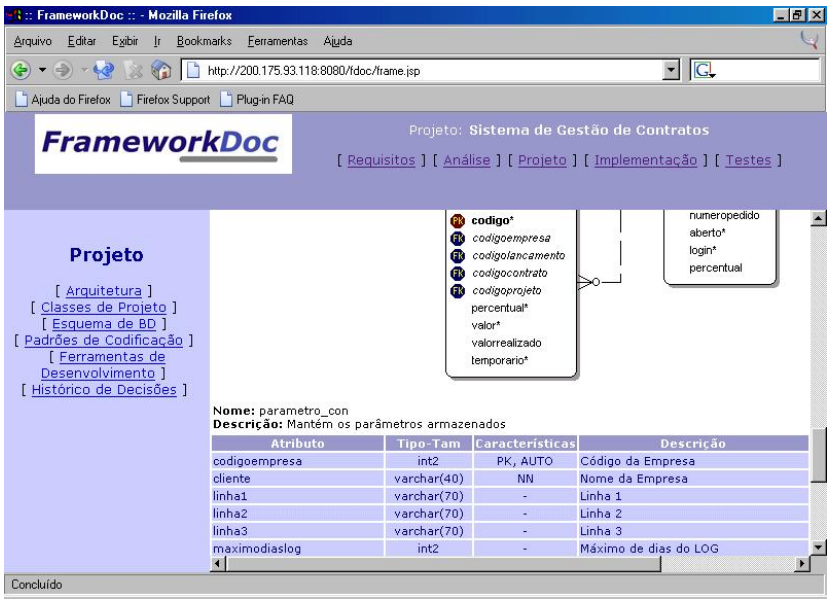

Figura 3.23: Esquema de Banco de Dados

Figura 3.24: Histórico de Decisões de Projeto

Na figura 3.25 é apresentado o menu *Implementação*, com a visualização do artefato *Documentação de Código*, integrada ao *JavaDoc*. O *FrameworkDoc* possui recursos que permitem a geração de outros artefatos de software, como mostrado na figura 3.26.

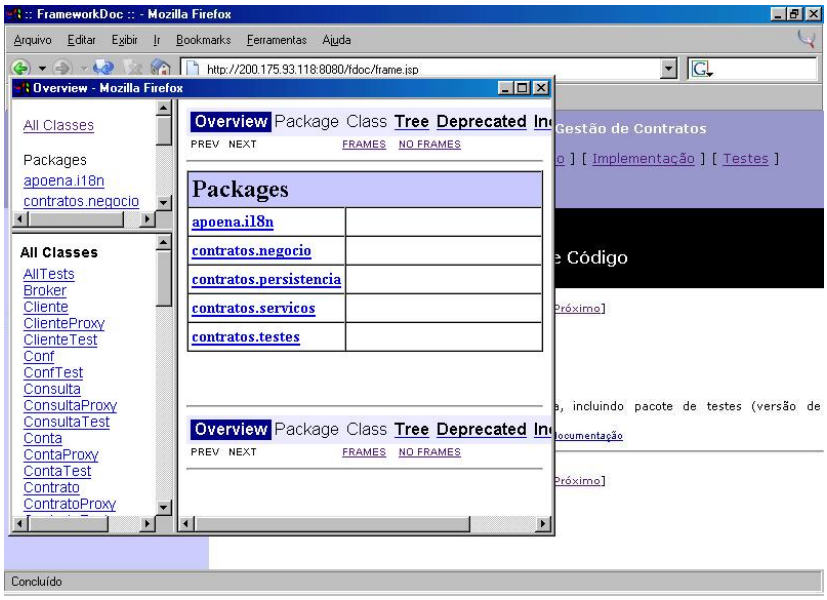

Figura 3.25: Documentação de Código

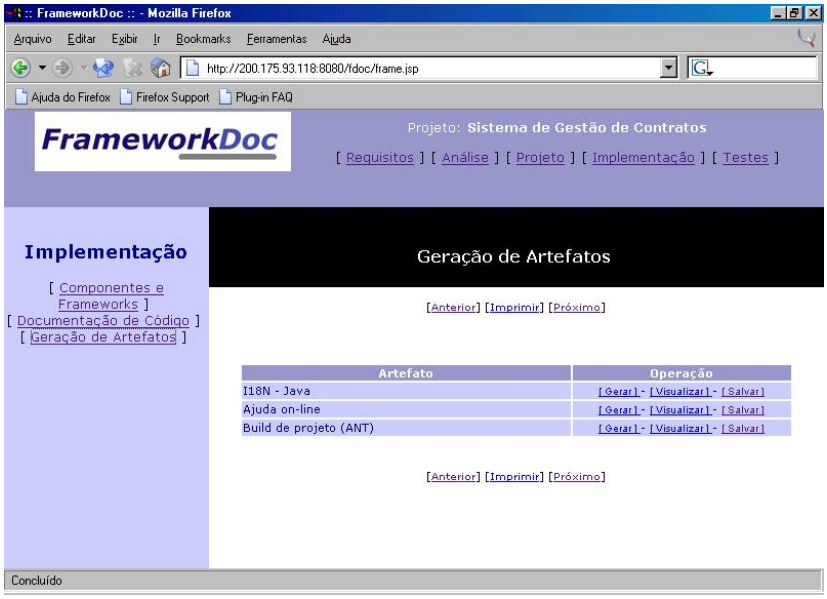

Figura 3.26: Geração de Artefatos

Para facilitar o trabalho em equipe faz-se necessário um conhecimento de todos os testes que devem ser realizados durante o desenvolvimento do software. Na figura 3.27, é apresentado o menu *Testes*, com a visualização do artefato *Testes de Unidade*, gerado a partir da definição do artefato *Casos de Uso Essenciais*.

Figura 3.27: Testes de Unidade

4 EXEMPLOS DE APLICAÇÃO DO FRAMEWORKDOC

Para consolidação e validação do *FrameworkDoc* como ferramenta de apoio ao desenvolvimento de software foram utilizados três projetos. Os sistemas desenvolvidos nestes projetos operam via *Web*, com algumas peculiaridades que são descritas a seguir. Os projetos foram desenvolvidos utilizando o PADS como processo de desenvolvimento e *frameworks* de serviços, componentes genéricos, relatórios e persistência desenvolvidos pela própria APOENA.

4.1 Descrição e Aplicação do *FrameworkDoc* em Projetos Reais

Os 3 projetos a seguir foram desenvolvidos pela equipe da APOENA para clientes reais. Por razões de confidencialidade, os nomes de alguns clientes e algumas informações essenciais foram mantidas em sigilo.

4.1.1 Sistema de Gestão de Contratos

O Sistema de Gestão de Contratos foi desenvolvido para uma empresa prestadora de serviços na área industrial. Através deste sistema, a empresa consegue gerenciar vários contratos e projetos através da *Web*.

O software permite o registro de todas as solicitações de material e despesas dos contratos, realizando o controle e acompanhamento do orçamento previsto. Além do registro, o sistema mantém um cadastro de plano de contas contábeis, clientes, contratos, projetos e orçamentos dos contratos. Com base neste sistema, é possível gerenciar os projetos/contratos específicos, com vários relatórios (formato HTML/PDF) e gráficos gerenciais, bem como fazer simulações de orçamento.

Para desenvolvimento deste sistema foi utilizado o servidor *Web Apache TomCat* com tecnologia *Java* (*Servlets*/JSP, *JavaBeans*) e SGBD *PostgreSQL*[34].

Foram um total de 23 *releases*, lançados quinzenalmente, que eram validados pelo cliente. Durante o período de desenvolvimento, houve um alto grau de envolvimento do

[34] Mais informações em http://www.postgresql.org

cliente com a equipe de desenvolvimento, principalmente na definição de prioridade de requisitos, validação do protótipo e definição dos testes.

O *FrameworkDoc* teve um papel decisivo no desenvolvimento e controle de artefatos do sistema, uma vez que centralizava todas estas informações de testes, histórico de decisões do projeto e, principalmente, definição de *layout* dos relatórios. À medida que o projeto avançava, eram disponibilizados mais artefatos técnicos e de apoio à equipe de desenvolvimento e ao cliente. Ao final do projeto, além de ter o software em funcionamento, o cliente teve a documentação técnica organizada e disponibilizada no seu servidor da aplicação.

4.1.2 Sistema de Controle Financeiro

O Sistema de Controle Financeiro, desenvolvido para uma empresa que possui várias filiais, permite gerenciar as contas bancárias e pagamentos de várias unidades, bem como um controle de contas a pagar e receber. Além disso, o sistema realiza um acompanhamento do orçamento anual, nas diversas rubricas.

O software também oferece diversas consultas como contas a pagar, extratos bancários, balancetes, acompanhamento do orçamento, livro caixa, entre outras (formato HTML/PDF). Foi desenvolvido um módulo de emissão de cheques *on-line*, permitindo que um usuário na *Web* consiga realizar a impressão de cheques em uma impressora específica.

Para desenvolvimento do módulo *Web*, foi utilizado o servidor *Web Apache*, linguagem de programação PHP e SGBD *PostgreSQL*. No módulo de impressão de cheques foi utilizado tecnologia *Java* (*Swing*, JDBC e JNI) acessando o mesmo Banco de Dados. Para tanto, foi desenvolvida uma DLL em C++ para fazer a ligação entre o módulo *Java* e a DLL da impressora de cheques.

O tempo de desenvolvimento deste projeto foi estimado em 3 meses, sendo que foi possível entregá-lo com 1 mês de antecedência. Neste período, foram elaborados vários protótipos para validação das opções e da interface gráfica da aplicação.

O *FrameworkDoc* foi utilizado como uma ferramenta de integração da documentação e dos artefatos, armazenando o histórico de decisões do projeto e listagem dos *releases*, facilitando o acompanhamento do cliente sobre as próximas versões. Como o cliente possui equipe própria de desenvolvimento, o *FrameworkDoc* facilitou muito o entendimento das regras de negócio e de outras questões técnicas.

4.1.3 Sistema FIEL Contábil

O Sistema FIEL Contábil foi elaborado a partir de uma demanda do Sindicato dos Bancários de Porto Alegre e Região. O cliente necessitava de uma solução que permitisse ao contador acompanhar a contabilidade da entidade, bem como fornecer mecanismos que permitissem um acompanhamento da execução orçamentária. A partir dessa demanda, foi elaborado o sistema de contabilidade e orçamento, intitulado FIEL Contábil.

O FIEL Contábil é um sistema desenvolvido com tecnologia de software livre que permite o registro e acompanhamento da contabilidade e da execução orçamentária de Sindicatos, Empresas e Entidades.

Na confecção do FIEL Contábil utilizou-se o servidor *Web Apache*, linguagem de programação PHP e o SGBD *PostgreSQL*.

O *FrameworkDoc*, além de centralizar as informações do projeto, permitiu documentar de forma eficiente as regras de negócio e o acompanhamento das decisões de projeto, principalmente relacionadas a interface gráfica. O FIEL Contábil é o primeiro software brasileiro licenciado sob *Creative Commons*[35] e está disponível para *download* em http://www.apoenasoftwarelivre.com.br/fielcontabil/.

4.2 Resultados

Durante a execução dos projetos pode-se constatar que, com a adoção do *FrameworkDoc* para documentação e geração de artefatos, houve um ganho significativo em vários aspectos como organização, agilidade, qualidade e produtividade. Os principais ganhos estão listados a seguir:

- Centralização das informações técnicas em uma única fonte, com publicação da documentação/geração de artefatos para a equipe;
- Ganho de tempo na documentação e criação de artefatos, com automação de processos, aumentando a qualidade e produtividade;
- Documentação criada e mantida durante todo o desenvolvimento do projeto, sem perda de informações relevantes;
- Melhoria da qualidade de apresentação da documentação, facilitando o acesso de todos os membros da equipe aos artefatos do projeto, mesmo que geograficamente distantes;
- Padronização na definição dos artefatos que são utilizados nos diversos projetos;
- Agregação de valor ao produto final.

4.3 Comparando com as outras ferramentas de documentação

Seguindo os mesmos critérios e características funcionais que são cobertas totalmente, parcialmente ou não cobertas pelas ferramentas descritas anteriormente, incluiu-se o *FrameworkDoc* na comparação.

[35] Mais informações em http://creativecommons.org/licenses/by-nc-sa/2.0/br/deed.pt

Tabela 4.1: Comparação entres as Ferramentas de Documentação, incluindo o *FrameworkDoc*

Critérios	***FrameDoc***	***EDoc***	***FrameworkDoc***
Exportação da documentação em formato portável	Exporta a documentação no formato HTML	Exporta a documentação no formato HTML	Exporta a documentação no formato HTML
Utilização de *hiperlinks* na documentação	Permite	Permite	Permite
Criação e configuração de padrões de documentação	Permite a criação de *templates* associados a categorias	Possui *templates* internos para documentação dos artefatos	Possui *templates* internos para documentação dos artefatos
Geração de documentação modular	Permite a geração de documentação de qualquer artefato de forma isolada	Ferramenta de integração, não permitindo a geração parcial da documentação	Permite a geração de documentação de qualquer artefato de forma isolada
Geração de documentação implícita ao artefato	Permite	Permite	
Tecnologias	*Java Swing*, *com* serialização, JDBC/ODBC e conexão com o GOA++[36]	Componentes COM (desenvolvidos em *Delphi*), ASP e SQL *Server*	Java (JSP, *Servlets*, *JavaBeans*), XML
Geração de Artefatos	Somente a documentação	Relatórios de requisitos, casos de uso e atores	Documentação, plano de testes, arquivo de *build* do *Apache ANT*, I18N para *Java*, *C* e *PHP*, ajuda da aplicação
Integração com outras ferramentas	Não	Integração com a *Rational Rose*	Integração com ferramentas de documentação de código (*JavaDoc*, *PHPDoc)*, *Apache ANT*
Dependência de Processos	Independente de processos	RUP	Independente de processo

[36] Mais informações em http://www.cos.ufrj.br/~goa/

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo realizar uma investigação, proposição e desenvolvimento de uma ferramenta de documentação e geração de artefatos de software, a partir de estudos relacionados ao uso de ferramentas, tecnologias, *frameworks*, documentação de software e geração de artefatos no processo de desenvolvimento de software.

No capítulo 2, foi realizado uma revisão de literatura sobre *frameworks*, documentação e geração de artefatos de software e processos de desenvolvimento OO com a finalidade de prover mecanismos de criação de uma ferramenta de documentação e geração de artefatos de software para a APOENA Software Livre. Também foi apresentado o *PADS – Processo APOENA de Desenvolvimento de Software*, com suas etapas, atividades e artefatos relacionados.

No capítulo 3, foi definida a estrutura, tecnologias e projeto da ferramenta desenvolvida, denominada *FrameworkDoc*. Quanto a estrutura, foram apresentados os componentes e os principais artefatos documentados e gerados pela ferramenta. Relacionado as tecnologias, foram listadas as tecnologias utilizadas para desenvolvimento do *FrameworkDoc*. No projeto, foi apresentada a arquitetura MVC com seus componentes, diagramas e padrões implementados.

Já no capítulo 4, a ferramenta desenvolvida foi testada em 3 estudos de caso reais, com peculiaridades diferentes, onde foram adaptadas algumas de suas características.

A documentação de software tem grande importância para o processo de desenvolvimento, principalmente por registrar decisões e requisitos que são fundamentais para o projeto. No caso da APOENA, por se tratar de uma empresa de desenvolvimento de software livre e *open-source*, a documentação se torna um produto que é entregue ao cliente, juntamente com o código-fonte. Portanto, a documentação é também um dos produtos que são desenvolvidos e assim a agilidade no desenvolvimento e na manutenção da documentação é essencial.

5.1 Contribuições

As principais contribuições deste trabalho são:

- Utilização de artefatos suficientemente genéricos que podem ser utilizados de forma individual ou integrada a outros processos de desenvolvimento, que utilizem a UML como linguagem de modelagem;

- Apresentação do PADS no 1º Congresso Brasileiro de Metodologias Ágeis de Software *(eXtreme Programming Brasil 2002*[37]*)*, focado na integração de práticas XP ao processo. A XP é uma disciplina relativamente nova, que mostrou-se muito importante principalmente na integração de equipes, fazendo com que elas trabalhem de forma mais cooperativa, sem perder o foco do projeto. O aumento de comunicação no processo, principalmente através da metáfora, sendo tratada dentro do PADS como um padrão de projeto de alto nível para estimativas de esforço, e da rotação dos membros da equipe trabalhando em todos os papéis, faz com que surja o que se denomina *interação extrema* (base de conhecimento comum dentro da empresa);
- Mais importante que realizar a documentação e geração de artefatos de um projeto de software, é o fato de centralizá-los e integrá-los em uma ferramenta, disponibilizando-os a qualquer momento a todos os componentes do projeto. Com o *FrameworkDoc* é possível disponibilizar os artefatos utilizados no projeto a todos os membros da equipe, através de uma estrutura *Web* (Intranet);
- Pelo fato de ser desenvolvido com tecnologias abertas e padrões conhecidos, tanto o PADS como o *FrameworkDoc* tem condições de serem integrados a outros processos e soluções de projetos de software, focados principalmente em agilidade, documentação e geração de artefatos.

5.2 Trabalhos Futuros

Através deste trabalho, é possível investigar as seguintes áreas:

- Criação de um *cookbook* ativo para utilização e instanciação do *FrameworkDoc*. Atualmente, o *FrameworkDoc* precisa ser instanciado por pessoas da equipe que possuam conhecimento em sua estrutura.
- Criação de um aplicativo de alimentação do *FrameworkDoc*, facilitando a entrada de informações e alimentação dos artefatos. Pelo fato dos artefatos serem armazenados em formato XML, os arquivos podem ser editados por qualquer editor de texto.
- Integração com conhecidas ferramentas de modelagem de software utilizadas no mercado, como *Rational Rose*, *MagicDraw*, *Posseidon*, entre outras, através de outras tecnologias como XMI[38]. Esta integração está facilitada justamente pelo *FrameworkDoc* ser desenvolvido com tecnologias abertas como o XML;
- Criação de *plugins* de conversão de artefatos, acoplando estes em IDEs de desenvolvimento como o *Eclipse, NetBeans* e ferramentas CASE como *Design For Databases*, entre outros;

---

[37] Realizado em dezembro de 2002, na FIAP, em São Paulo. Mais Informações em http://www.xispe.com.br/evento2002/

[38] *XML Metadata Interchange.* Mais informações em http://www.omg.org/technology/xml/index.htm

- Investigação de outros formatos de documentos, como PDF e RTF, para exportação da documentação gerada pelo *FrameworkDoc;*

Controle de versões de documentação e geração de artefatos de software. Atualmente, o *FrameworkDoc* mantém a documentação em um estado único, o atual.

REFERÊNCIAS

AHMED, K. Z.; UMRYSH, C. **Desenvolvendo Aplicações Comerciais em Java com J2EE e UML.** [S.l.]:Ciência Moderna, 2002.

ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE Patterns:** as melhores práticas e estratégias de design. Rio de Janeiro: Campus, 2002.

AMBLER, S. **Uma visão realística da reutilização em Orientação a Objetos.** Disponível em: <http://www.uml.com.br/arquivos/tutoriais/umavisaorealisticadareutilizacao.doc.> Acesso em: jun. 2002.

AMBLER, S. **Modelagem Ágil:** práticas eficazes para a Programação eXtrema e o Processo Unificado. Porto Alegre: Bookman, 2004.

APACHE Jakarta TomCat. Disponível em: <http://jakarta.apache.org/tomcat/index.html>. Acesso em: set. 2004.

APACHE ANT Welcome**.** Disponível em: <http://ant.apache.org>. Acesso em: set. 2004.

ASTELS, D.; MILLER, G.; NOVAK, M. **eXtreme Programming:** guia prático. Rio de Janeiro: Campus, 2002.

BECK, K. **eXtreme Programming Explained.** [S.l.]: Addison-Wesley, 2000.

BECK, K.; FOWLER, M. **Planning eXtreme Programming.** [S.l.]: Addison-Wesley, 2001.

BOOCH, G. **Object-Oriented Analysis and Design.** Redwood City: Benjamin/Cummings, 1994.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**: guia do usuário. Rio de Janeiro: Campus, 2000.

BORGES, M. **Tópicos Especiais em Sistemas de Informação.** 1998. Notas de Aula do Curso, DCC/UFRJ.

BRAGA, R.; MASIERO, P. C. Identification of Framework hot-spots using Patterns Languages**.** In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 15., 2001, Rio de Janeiro. **Anais...** Rio de Janeiro: UFRJ, 2001.

BROGDEN, B.; MINNICK, C. **Guia do Desenvolvedor Java:** desenvolvendo E-Commerce com Java, XML e JSP. [S.l.]: Makron Books, 2002.

BUSCHMANN, F. The Master-Slave Pattern**.** In: CONFERENCE ON PATTERN LANGUAGES AND PROGRAMMING, 1., 1994. **Proceedings...** [S.l.:s.n.], 1994.

BUSCHMANN, F. et al. **Pattern-Oriented Software Architecture:** a system of patterns. [S.l.]: John Wiley & Sons, 1996.

COLEMAN, D. et al. **Object-Oriented Development:** the fusion method. Englewood Cliffs, N.J.: Prentice Hall, 1994.

CRESPO, S. **Composição em WebFrameworks.** 2000. Tese (Doutorado), PUC-Rio.

CUNNINGHAM, W. **Extreme Programming.** Disponível em: <http://www.c2.com/cgi/wiki? ExtremeProgramming>. Acesso em: jun. 2002.

DÉCIO, O. **XML – Guia de Consulta Rápida.** [S.l.]: Novatec, 2000.

DEITEL, H.; DEITEL, P. **Java como Programar.** 3. ed. Porto Alegre: Bookman, 2000.

ECKEL, B. **Thinking in Java**. [S.l.:s.n.], 1997.

FAYAD, M.; SCHMIDT, D.; JOHNSON, R. **Building Application Frameworks:** object-oriented foundations of framework design. [S.l.]: John Wiley & Sons, 1999.

FIELDS, D.; KOLB, M. **Desenvolvendo na Web com JavaServer Pages.** [S.l.]: Ciência Moderna, 2000.

FILETO, R.; MEIRA, C.; COSTA, C.; MASSHURÁ, S. A Construção de um Gerador de Programas Aplicativos segundo Conceitos de Análise de Domínio**.** In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 10., 1996. **Anais...** [S.l.:s.n.], 1996.

FONTOURA, M. F. **Uma Abordagem Sistemática para o Desenvolvimento de Frameworks.** 1999. Tese (Doutorado), PUC-Rio.

FOWLER, M. **Refactoring:** improving the design of existing code. [S.l.]: Addison-Wesley, 1999.

FOWLER, M.; SCOTT, K. **UML Essencial:** um breve guia para a Linguagem Padrão de Modelagem de Objetos. Porto Alegre: Bookman, 2000.

FRANCA, L. P.; STAA, A. V. Geradores de Artefatos: Implementação e Instanciação de Frameworks**.** In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 15., 2001, Rio de Janeiro. **Anais...** Rio de Janeiro: UFRJ, 2001.

FURLAN, J. D. **Modelagem de Objetos através da UML:** análise e desenho orientados a objetos. [S.l.]: Makron Books, 1998.

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

GHEZZI, C.; JAZAYERI, M.; MANDRIOLI, D. **Fundamentals of Software Engineering**. [S.l.]: Prentice Hall, 1991.

GOMES, A.; OLIVEIRA, K.; ROCHA, A. R. Avaliação de Processos de Software baseada em Medições**.** In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 15., Rio de Janeiro. **Anais...** Rio de Janeiro: UFRJ, 2001.

HATCHER, E.; LOUGHRAN, S. **Java Development with ANT.** [S.l.]: Makron Books, 2003.

HUNT, A.; THOMAS, D. **Pragmatic Unit Testing in Java with JUnit.** [S.l.]: The Pragmatic Bookshelf, 2003.

JACOBSON, I. et al. **Object-Oriented Software Enginnering:** a use case driven approach. [S.l.]: Addison-Wesley, 1992.

JACOBSON, I.; RUMBAUGH, J.; BOOCH, G. **The Unified Software Development Process.** [S.l.]: Addison-Wesley, 1999.

JAVA Logging API. Java.util.logging (Java 2 Platform SE v. 1.4.2). Disponível em: <http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html>. Acesso em: jul. 2004.

JEFFRIES, R.; ANDERSON, A.; HENDRICKSON, C. **eXtreme Programming Installed.** [S.l.]: Addison-Wesley, 2001.

JOHNSON, R.; FOOTE, B. Designing Reusable Classes. **Journal of Object-Oriented Programming**, 1988.

JOHNSON, R.; RUSSO, V. F. **Reusing Object-Oriented Design.** [S.l.]: University of Illinois, 1991. Technical Report.

JOHNSON, R. Documenting Frameworks using Patterns. **SIGPLAN Notices**, New York, v. 27, n. 10, Oct. 1992. Trabalho apresentado na Conference on Object-Oriented Programming Languages and Applications,1992, Vancouver, CA.

JÚNIOR, F. B. **JSP:** A Tecnologia Java na Internet. São Paulo: Érica, 2002.

JUnit, Testing Resources for Extreme Programming. Disponível em: <http://www.junit.org>. Acesso em: set. 2004.

LARMAN, C. **Utilizando UML e Padrões:** uma introdução à análise e ao projeto orientado a objetos. Porto Alegre: Bookman, 2000.

LARMAN, C. **Craig Larman´s Home Page.** Disponível em: <http://www.craiglarman.com>. Acesso em: set. 2003.

LOBO, S.; OLIVEIRA, R.; TELES, V. EDoc: a documentation tool for UML based models. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 15., 2001, Rio de Janeiro. **Anais...** Rio de Janeiro: UFRJ, 2001.

MARTIN, J.; ODELL, J. **Object-Oriented Methods:** a foundation. [S.l.]: Prentice Hall, 1995.

MARTIN, R. **RUP vs. XP.** Object Mentor Developer Resources. Disponível em: <http://www.objectmentor.com/resources/articles/RUPvsXP.pdf>. Acesso em: ago. 2002.

MATOS, A. **UML:** Prático e Descomplicado. [S.l.]: Érica, 2002.

MATTSSON, M. **Evolution and Composition Object-Oriented Frameworks.** 2000. PhD Thesis. University of Karlskrona.

MURTA, L. G. P. **FrameDoc:** um framework para documentação de componentes reutilizáveis. 1999. Trabalho de Diplomação, DCC/UFRJ. Rio de Janeiro.

OBJECT DESIGN. **Larman´s UML Process.** Disponível em: <http://www.objectsbydesign.com/books/ larman_process.html>. Acesso em: set. 2003.

OBJECT MENTOR. **Object Mentor – Extreme Programming.** Disponível em: <http://www.objectmentor.com/processImprovement/index>. Acesso em: jun. 2002.

OBJECT MENTOR. **Pair Programming, an eXtreme Programming practice.** Disponível em: <http://www.pairprogramming.com>. Acesso em: maio 2004.

ORTIGOSA, Á. M. **Proposta de um Ambiente Adaptável de Apoio ao Processo de Desenvolvimento de Software.** 1995. Dissertação (Mestrado em Ciência da Computação), UFRGS, Porto Alegre.

PREE, W. **Design Patterns for Object-Oriented Software Development.** Workinggham: Addison-Wesley, 1995.

PRESSMAN, R. **Engenharia de Software**. São Paulo: Makron Books, 1995.

RUMBAUGH, J. et al. **Object-Oriented Modeling and Design.** Englewoods Cliffs: Prentice Hall, 1991.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual.** Reading, MA: Addison-Wesley, 1999.

SILVA, R. P. **Suporte ao desenvolvimento e uso de Frameworks e Componentes.** 2000. Tese (Doutorado em Ciência da Computação), UFRGS, Porto Alegre.

SILVA, O. J. **XML:** Aplicações Práticas. [S.l.]: Érica, 2001.

SMITH, J. **A Comparison of RUP and XP.** Rational Software White Paper. Disponível em: <http://www.rational.com/media/whitepapers/TP167.pdf>. Acesso em: maio 2004.

SUN MICROSYSTEMS. **The Source for Java Technology.** Disponível em: <http://java.sun.com>. Acesso em: maio 2004.

TELES, V. **eDoc.** Disponível em: <http://tecodonte.nce.ufrj.br/uml/aulas/2001/aulas2/eDoc/index.htm>. Acesso em: set. 2002.

TREVISAN, A. R. **Framework para Integração de Ferramentas.** 1994. Trabalho Individual (Mestrado em Ciência da Computação), UFRGS, Porto Alegre.

VELOSO, R. **Java e XML - Processamento de documentos XML com Java:** guia de consulta rápida. [S.l.]: Novatec, 2003.

WEINAND, A.; GAMMA, E.; MARTY, R. **Design and Implementation of ET++, a Seamless Object-Oriented Application Framework. Structured Programming**, [S.l.], v. 10, n. 2, p. 63-87, Abril 1994.

WIRFS-BROCK, R.; JOHNSON, R. Surveying Current Research in Object Design. **Communications of the ACM**, [S.l.], 1990.

WUESTEFELD, Klaus. **Xispê.** Disponível em: <http://www.xispe.com.br>. Acesso em: maio 2004.

ZANCAN, J. C. **Um Sistema em Hiperdocumentos para Apoio à Instanciação de Frameworks.** 1999. Dissertação (Mestrado em Ciência da Computação), UFRGS, Porto Alegre.