UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DISSERTAÇÃO DE MESTRADO

Apoio de Gerência de Configuração de Software ao ARA-GAwCRe

SIMONE DE SOUSA BORGES

São Carlos/SP
Agosto/2008

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

B732ag

Borges, Simone de Sousa.
Apoio de gerência de configuração de software ao ARA-GAwCRe / Simone de Sousa Borges. -- São Carlos : UFSCar, 2009.
102 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2008.

1. Gerência de configuração de software. 2. Reengenharia de software. 3. Melhoria de processo. 4. Geradores de aplicação. I. Título.

CDD: 005.30288 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Apoio de Gerência de Configuração de Software ao ARA-GAwCRe"

SIMONE DE SOUSA BORGES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa.Dra. Rosângela Ap. Dellosso Penteado
(Orientadora - DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado
(DC/UFSCar)

Profa. Dra. Elisa Hatsue Moriya Huzita
(UEM)

São Carlos
Agosto/2008

*A meus Pais (*in memoriam*).*

Agradecimentos

A Deus pela oportunidade e por ter me concedido forças para trilhar este caminho.

À Prof[a]. Dr[a]. Rosângela, pela confiança, amizade e paciência que me dedicou. Meu muito obrigada pela orientação.

Ao meu amigo Vinícius Durelli, pela companhia, pelos conselhos e pelas revisões. Por todos os momentos tristes e alegres que passamos em São Carlos.

Ao meu amigo Jerônimo Pereira, por todo o apoio e por ter sempre uma palavra amiga a dizer quando eu mais precisava.

À Prof[a]. Dr[a]. Rosely Sanches e ao Prof. Dr. Antônio F. do Prado pelos conselhos valiosos.

À minha irmã Adriana, meu cunhado Sebastião e meu amado sobrinho Bruno. Obrigada por sempre me receberem de braços abertos.

A Débora, Lidiane e Marisa, companheiras de república, por todos os momentos que rimos e choramos, pelas conversas e pelo incentivo mútuo.

"Talento, vocação, amor e desejo não são suficientes para fazer um projeto dar certo. É preciso uma preparação adequada."
— Roberto Shinyashiki

"As únicas coisas que evoluem por vontade própria em uma organização são a desordem, o atrito e o mau desempenho."
— Peter Drucker

Resumo

Sistemas de software passam por inúmeras modificações o longo de seu ciclo de vida. Gerência de Configuração de Software (GCS) preocupa-se em organizar, controlar e gerenciar o desenvolvimento e a evolução desses sistemas. GAwCRe é um gerador de aplicações para a Web desenvolvido com base em Linha de Produtos de Software, no domínio de clínicas de reabilitação física (Fisioterapia, Terapia Ocupacional e Educação Física). A instanciação do gerador é feita a partir de uma Linguagem de Modelagem de Aplicação definida com base na linguagem de padrões SiGCli. Por um processo ad hoc foi proposta a utilização do gerador GAwCRe como ferramenta de apoio junto ao Arcabouço de Reengenharia Ágil (ARA). ARA apóia a migração de sistemas legados procedimentais para o paradigma orientado a objetos utilizando recursos como: reúso de projeto e de código, processos, linguagens de padrões de análise, práticas de métodos ágeis e ferramentas. Esta dissertação apresenta uma abordagem de GCS junto ao ARA para apoiar o Processo de Reengenharia Ágil de Sistemas Legados Utilizando Geradores de Aplicações. A abordagem proposta utiliza processo baseado na norma IEEE 828-2005 para guiar o planejamento das atividades de GCS. Foram empregados três sistemas: controle de versões, controle de mudanças e gerenciamento de construções. Outros sistemas e ferramentas são previstos para prover a integração dos espaços de trabalho. A abordagem proposta apóia o emprego do Processo Ágil de Reengenharia Utilizando Geradores de Aplicações em associação ao ARA, assim, outros geradores de aplicações construídos com base em linguagens de padrões podem ser utilizados para auxiliar a obtenção do projeto e para a implementação do sistema alvo. A reengenharia de dois sistemas legados foi parcialmente realizada apoiada pela abordagem proposta.

Abstract

Software systems suffer countless modifications during all its life cycle. Software Configuration Management (SCM) is concerned with organizing, controlling and managing the development and evolution of those systems. GAwCRe is a web application generator built based on Software Product Line for the rehabilitation clinics management domain. The generator instantiation is made through a Modelling Language that is based on the pattern language SiGCli. An ad hoc process has been proposed in order to apply the GAwCRe application generator within *Arcabouço de Reengenharia Ágil* (ARA). The ARA has the purpose of migrating small and medium procedural systems to the OO paradigm. This research presents an approach of SCM along with ARA to support the *Processo de Reengenharia Ágil de Sistemas Legados Utilizando Geradores de Aplicações*. The proposed approach uses a process, that is based on the standard IEEE 828-2005, to guide the planning of the SCM activities. Three systems were used: version control, changes control and build management. Other systems and tools are foreseen to provide the integration of the work spaces. The proposed approach supports the employ of the *Processo Ágil de Reengenharia Utilizando Geradores de Aplicações* associated with the ARA, other generators of applications built based in pattern languages also can be used to aid the design and implementation of the resulting system. Two legacy systems were partially reengineered applying the proposed approach.

Sumário

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Abreviaturas

ARA: Arcabouço de Reengenharia Ágil
CMMI: *Capability Maturity Model Integration*
CSS: *Cascading Style Sheets*
GAwCRe: Gerador de Aplicações baseadas na Web para sistemas de gestão de Clínicas de Reabilitação
GCS: Gerência de Configuração de Software
GDMS: Grupo de Desenvolvimento e Manutenção de Software
GREN: Framework para Gestão de Recursos de Negócios
GRN: Linguagem de Padrões para Gestão de Recursos de Negócios
IC: Item de Configuração
IDE: *Integrated Development Environment*
IEC: *International Electrotechnical Commission*
IEEE: *Institute of Electrical and Electronics Engineers*
ISO: *International Organization for Standardization*
LMA: Linguagem de Modelagem de Aplicação
LPA: Linguagem de Padrões de Análise
LPS: Linha de Produtos de Software
MPS-BR: Melhoria de Processos do Software Brasileiro
PARFAIT: Processo Ágil de Reengenharia baseado em Framework no domínio de sistemas de Informação com técnicas de Validação, Verificação e Teste
PGCS: Plano de Gerência de Configuração de Software
PL: *Persistent Layer*
RB: *Release Branch*
RUP: *Rational Unified Process*
SCM: Sistema de Controle de Mudanças
SCV: Sistema de Controle de Versões
SGBD: Sistema de gerenciamento de banco de dados
SiGCli: Linguagem de Padrões para Sistemas de Gerenciamento de Clínicas de Reabilitação
SPICE: *Software Process Improvement and Capability dEtermination*
SQL: *Structured Query Language*
Std: *Standard*
SVN: Subversion
UML: *Unified Modeling Language*
VV&T: Validação, Verificação & Testes
XML: *EXtensible Markup Language*

CAPÍTULO
1

Introdução

Sistemas de software passam por uma série de modificações ao longo de seu ciclo de vida. Gerência de Configuração de Software (CGS) preocupa-se em estabelecer critérios que permitam a realização de tais modificações, todavia assegurando a consistência e a integridade dos sistemas de software e também a aderência aos requisitos. GCS pode ser definida como a disciplina que permite manter sob controle a evolução dos sistemas de software, gerenciando e rastreando mudanças e restrições de qualidade e cronograma (Estublier, 2000).

Durante o desenvolvimento de projetos de software, diversos itens são criados e modificados. Inicialmente as pesquisas sobre GCS tinham enfoque no armazenamento de código fonte (Murta, 2006), posteriormente buscou-se prover recursos para outros níveis de abstração como análise e projeto. GCS é considerada uma disciplina de engenharia de software madura e importante para a garantia da qualidade do processo de desenvolvimento de software.

Abordagens de desenvolvimento baseadas em Linha de Produtos de Software (LPS) são cada vez mais utilizadas por organizações que necessitam desenvolver diferentes variantes de seus produtos. O reúso obtido pelo emprego de LPS no desenvolvimento de software facilita a criação de aplicações, pois os desenvolvedores não precisam iniciar do zero, com isso o prazo de desenvolvimento pode tornar-se menor. LPS consiste em uma família de sistemas de software que possui um núcleo de funções em comum, cuja vantagem é formar um conjunto de recursos reusáveis e funções variáveis, que permite a especialização das aplicações para atender a domínios específicos (Pohl et al., 2005; Gomaa e Shin, 2007). Tanto as funções em comum, quanto as variáveis podem evoluir de modo independente, por esse motivo o apoio da GCS é ainda

mais importante no desenvolvimento de sistemas utilizando essa abordagem.

O Arcabouço de Reengenharia Ágil (ARA) (Cagnin, 2005) foi modificado para que geradores de aplicações fossem utilizados no lugar de frameworks. ARA apóia a reengenharia de sistemas legados fornecendo, em um curto espaço de tempo, produtos com qualidade utilizando recursos como metodologias ágeis (Canfora e Penta, 2007) e linguagem de padrões de análise (Cagnin et al., 2003). Freitas (2006) realizou um estudo de caso em que o gerador de aplicações GAwCRe (Pazin, 2004) foi utilizado junto ao ARA. Esse gerador foi construído com base em LPS. Dessa forma, ARA com gerador de aplicações GAwCRe pode ser utilizado tanto para o desenvolvimento de sistemas, quanto para a realização de reengenharia. Em ambos os casos, a GCS é necessária para controle efetivo dos produtos gerados.

1.1 Objetivo

O principal objetivo deste trabalho é: apresentar uma abordagem de GCS junto ao ARA (Cagnin, 2005) para apoiar o Processo de Reengenharia Ágil de Sistemas Legados Utilizando Geradores de Aplicações (Freitas, 2006), uma vez que nenhum controle de versões é possível junto aos produtos gerados.

A abordagem proposta é baseada em processo, sistemas e ferramentas de apoio. O processo elaborado é baseado na norma IEEE 828-2005 (IEEE, 2005) que define um plano de GCS que prevê customizações em face às necessidades do projeto. Para automatizar a execução das atividades que compõem o processo de GCS são empregados três sistemas: controle de versões, controle de mudanças e gerenciamento de construções; e para que esses sistemas possam ser integrados e efetivamente utilizados outros serviços são disponibilizados para prover a integração dos espaços de trabalho (Murta, 2006). A abordagem proposta apóia o emprego do Processo Ágil de Reengenharia Utilizando Geradores de Aplicações em associação ao ARA, assim, geradores de aplicações construídos com base em linguagens de padrões podem ser utilizados para auxiliar a obtenção do projeto e para a implementação do sistema alvo.

1.2 Justificativa

ARA (Cagnin, 2005) apóia a migração de sistemas legados procedimentais para o paradigma orientado a objetos por meio de recursos como: reúso de projeto e de código, processos, linguagens de padrões de análise, práticas de métodos ágeis e ferramentas. Tal conjunto de recursos é utilizado com o objetivo de se reduzir o tempo, os custos e também os riscos de reengenharia (Rosenberg, 1996). Originalmente ARA foi utilizado com o processo PARFAIT (Cagnin, 2005) que utiliza frameworks para a instanciação de sistemas. A necessidade de controle de versões, no contexto de frameworks é res-

saltada por (Cagnin, 2005) tanto para controlar as versões do framework quanto as das aplicações criadas por meio de sua instanciação. Assim, a ferramenta GREN-WizardVersionControl (Cagnin, 2005) foi elaborada para realizar o controle de versões das aplicações criadas pelo PARFAIT usando o framework GREN (Braga, 2002).

O Processo Ágil de Reengenharia Utilizando Geradores de Aplicações (Freitas, 2006) é uma adaptação do processo PARFAIT no qual o framework GREN foi substituído pelo gerador GAwCRe (Pazin, 2004). Em um estudo de caso realizado por Freitas (2006), a reengenharia de sistemas legados pertencentes a domínios mais amplos ao qual o gerador foi desenvolvido, acarretou modificações para atender essa ampliação de domínio. Sucessivas alterações na especificação do gerador foram realizadas sem controle de versões, todavia, no desenvolvimento e na manutenção de linha de produtos de software, o apoio de atividades de GCS é recomendado a fim de evitar sua degradação (Schäfer, 1996; Staples, 2004; Yu e Ramaswamy, 2006). O desenvolvimento de LPS tem se consolidado em muitas organizações como opção de criação de variantes diferentes para seus produtos. Um obstáculo a essa consolidação reside no apoio que sistemas de GCS atuais provêem para LPS, considerado ainda incipiente (Krikhaar e Crnkovic, 2007) tendo em vista a complexidade em se controlar variabilidades e similaridades inerentes a essa abordagem de desenvolvimento (Linden et al., 2007; Thao et al., 2008).

1.3 Metodologia

Para a realização deste trabalho as seguintes etapas foram seguidas: elaboração do referencial teórico, definição da abordagem de GCS proposta, aplicação da abordagem de GCS e elaboração das considerações finais.

Elaboração do referencial teórico: foi realizada pesquisa bibliográfica e as principais fontes empregadas são citadas nas referências deste trabalho. Essa etapa forneceu o estado da arte para a pesquisa, a fundamentação para a proposta do trabalho e também as definições utilizadas.

Definição da abordagem de GCS proposta: o principal referencial empregado para elaboração da abordagem foi a norma IEEE 828-2005 (IEEE, 2005) que guiou a definição do processo de GCS. Também foram consideradas as características ágeis da abordagem ARA a fim de não comprometer sua utilização.

Aplicação da abordagem de GCS: para avaliação da abordagem proposta, estudos de caso que consistem da reengenharia de dois sistemas legados foram realizados. As atividades do Processo de Reengenharia Ágil Utilizando Geradores de Aplicações foram apoiadas pela abordagem de GCS proposta neste trabalho.

Elaboração das considerações finais: A conclusão do trabalho é baseada na avaliação da abordagem proposta, considerando os resultados do estudo de casos e do referencial teórico pesquisado. Alguns trabalhos futuros também são indicados para complementar este.

1.4 Organização do Texto

Este trabalho está dividido em sete capítulos. O Capítulo 1 contextualiza a proposta deste trabalho apresentando a motivação para realizá-lo.

No Capítulo 2 apresenta-se uma introdução à disciplina de GCS, na qual são descritos suas perspectivas, funções e os sistemas e ferramentas de apoio que viabilizam sua execução.

O Capítulo 3 descreve abordagens que podem ser usadas para a realização de reengenharia de sistemas legados utilizando geradores de aplicações construídos com base em LPS, padrões de análise e abordagem ágil que utiliza frameworks. Inicialmente, descreve-se o gerador GAwCRe, a origem do projeto e seu emprego em um processo ágil de reengenharia associado à abordagem ARA, bem como os demais trabalhos relacionados a este.

No Capítulo 4 a abordagem de GCS proposta é apresentada e sua elaboração detalhada. No Capítulo 5 apresenta-se um estudo de casos em que sistemas legados passam por reengenharia utilizando ARA com gerador de aplicações e GCS. Para que o gerador GAwCRe fosse utilizado, algumas tarefas de manutenção corretiva e perfectiva foram executadas utilizando o controle de GCS.

O Capítulo 6 abstrai o processo ágil de reengenharia para geradores de aplicações com GCS e define um modelo para a apresentação das atividades que compõem as fases do Processo Ágil de Reengenharia Utilizando Geradores de Aplicações e GCS de forma que essas atividades possam ser usadas com outros geradores de aplicações construídos com base em linguagens de padrões.

No Capítulo 7 são comentadas as conclusões, as contribuições deste trabalho, relatam-se as limitações detectadas e são enumeradas sugestões de possíveis trabalhos futuros.

CAPÍTULO
2

Gerência de Configuração de Software

2.1 Considerações Iniciais

Gerência de Configuração de Software (GCS) representa um conjunto de atividades cujo objetivo é apoiar o processo de software durante seu ciclo de vida, de forma que as mudanças inerentes ao processo sejam assimiladas sem desestabilizá-lo (Pressman, 2006). Durante o processo de desenvolvimento de software, mudanças são inevitáveis, mudam-se os requisitos, as regras de negócio, necessidades podem ser revistas e alteradas. Assim, a importância da GCS está em manter a iteratividade entre as mudanças e os elementos que compõem o projeto em um estado de não degradação (Hass, 2003). Entre as atividades necessárias para se alcançar esses objetivos estão: a documentação e o armazenamento de informações e artefatos considerados pertinentes em um projeto, provendo meios de se recuperar, manusear e alterar informações de forma controlada e mantendo a integridade do projeto durante sua evolução; examinar, aprovar e controlar mudanças; capacidade de rastrear e auditar as modificações já realizadas (IEEE, 2005).

A GCS é considerada uma disciplina complexa, mas sendo uma atividade de garantia da qualidade de software (Pressman, 2006), é tão importante a ponto de modelos de maturidade de processo de desenvolvimento tais como SPICE (ISO, 2004), MPS-BR (SOFTEX, 2007) e CMMI (Ahern et al., 2008) a considerarem fundamental para a sua realização.

GCS provê visibilidade ao estado de evolução de um produto de software, benefici-

ando desenvolvedores, testadores de software, gerentes de projeto e demais envolvidos no processo. Entre os benefícios estão: a organização de tarefas e atividades que proporcionam a integridade do produto de software; a habilidade de rastrear mudanças; assegurar a correta configuração do software; assegurar que tarefas de implementação ocorram na versão correta; garantir a entrega da versão correta aos clientes; redução no ciclo de vida de manutenções; possibilidade de rastrear responsabilidades por eventuais mudanças; favorece um ambiente estável, controlado, reproduzível e passível de ser melhorado; elevar a conformidade na aplicação de normas; entre outros benefícios (Keyes, 2004). A seguir, alguns termos importantes para compreensão deste trabalho, relacionados à GCS, serão apresentados.

2.1.1 Item de Configuração e Metadados

Um item de configuração (IC) é, geralmente, a unidade básica de informação controlada em um processo de GCS. Qualquer artefato envolvido no ciclo de vida do sistema pode ser controlado, como por exemplo, código-fonte, componentes, relatórios diversos, documentos resultantes da análise e modelagem, até mesmo hardware e ferramentas.

A escolha dos artefatos que serão considerados itens de configuração e, por conseguinte estarão sob o controle da GCS, é uma decisão de projeto, e não do plano de GCS, e depende unicamente da necessidade em se controlar ou não esses artefatos, adequando-se a granularidade à norma ou ao modelo de processo que estiver sendo utilizado. Projetos de alto risco podem possuir um alto grau de formalismo na definição de quais serão os ICs devido à necessidade de controle inerente a este tipo de projeto. Em projetos não tão complexos, pode-se optar por combinar um baixo grau de formalismo em atividades mais simples, e aplicar um formalismo maior em atividades críticas como, por exemplo, manutenção e entrega (Hass, 2003).

Todas as informações a respeito de um IC são chamadas metadados. Metadados podem conter o nome do IC, sua data de criação, o local onde está armazenado, quem o criou, todo o tipo de informação considerada relevante no contexto onde o IC está inserido. A quantidade e o nível de complexidade das informações armazenadas para cada IC, dependerá também das necessidades do projeto. Uma quantidade menor de metadados provê facilidade na manutenção dessas informações, entretanto em algum momento podem ser insuficientes na resolução de problemas ou tomada de decisões, por outro lado um processo exaustivamente documentado pode tornar-se difícil de ser gerenciado, tal o nível de burocracia que pode atingir.

2.1.2 *Baseline*, *Release*, Revisão e Versão

Na indústria, por vezes, ocorre o uso indiscriminado de termos, como versão e *baseline* que terminam por confundir os envolvido no processo de GCS (Hass, 2003). Com o propósito de se utilizar um vocabulário coerente às práticas de GCS, algumas definições encontradas na literatura foram utilizadas e são apresentadas.

A definição do IEEE para *baseline* diz se tratar de uma especificação ou produto, formalmente revisto e aprovado, que servirá como base para futuros desenvolvimentos, e que eventuais modificações só ocorrerão sob procedimentos formais de controle de modificações (IEEE, 1990). O conceito de *baseline* envolve o ciclo de vida de um IC. Um artefato tomado como IC pode ser revisto e modificado inúmeras vezes, entretanto após ser incorporado a uma *baseline*, toda e qualquer modificação solicitada para o IC só irá acontecer após um processo formal de análise e aprovação da modificação. O propósito é justamente manter a integridade dos ICs, o que torna essa atividade um dos pilares da GCS.

Um *release* de software representa uma distribuição de uma baseline. Releases são classificados de acordo com o tipo de distribuição. Um *release alpha* é uma versão em desenvolvimento que foi disponibilizada somente a envolvidos no processo com fins de homologação. Um *release beta* é uma primeira versão externa de um software, liberada para testes e avaliação. Um *release candidate* é uma versão com todas as funções projetadas, já depurada e aparentemente livre de erros, e candidata a entrar em produção.

Cada vez que uma operação de *commit* é realizada, para persistir alterações realizadas em um IC, o estado do repositório onde estão armazenados esses ICs é alterado e uma nova versão do IC é criada. Na indústria, o termo versão pode ser encontrado referenciando um *release* específico de um produto de software, por este motivo alguns sistemas de controle de versões preferem utilizar o termo revisão para se referir às sucessivas alterações que ocorrem no ciclo de vida de um IC (Louridas, 2006; Collins-Sussman et al., 2007). Neste trabalho, sempre que necessário será empregado o termo versão no mesmo sentido empregado na norma IEEE std 828-2005 que o utiliza sempre que necessário se referir a uma evolução em um IC, independente se a intenção dessa evolução caracteriza-se como uma revisão ou uma variante, que são subclassificações para o termo versão (IEEE, 1987) (Conradi e Westfechtel, 1997).

2.1.3 Perspectivas em Gerência de Configuração de Software

As perspectivas em relação a GCS variam de acordo com o papel exercido pelos envolvidos no processo de software (Asklund e Bendix, 2002). Na Perspectiva Gerencial, as práticas de GCS são consideradas sob o ponto de vista do gerenciamento de um projeto, no qual, controle, métricas e avaliações são necessidades constantes. O seu

enfoque está no processo de desenvolvimento do produto de software, e o objetivo principal é a identificação e a documentação dos artefatos que compõem o produto controlando continuamente as modificações (Asklund e Bendix, 2002). A norma IEEE 828 divide essa perspectiva em cinco atividades:

- Identificação da configuração: O objetivo dessa atividade é determinar os metadados de um item de configuração, identificando-o, suas versões e também seu relacionamento com outros itens de configuração (Hass, 2003);
- Controle da configuração:
- Relato da situação da configuração;
- Avaliação e revisão da configuração;
- Gerenciamento de liberação e entrega.

A segunda perspectiva considera as necessidades no contexto de Desenvolvimento, enfocando espaço de trabalho, código-fonte, versionamento, entre outros, entre seus objetivos estão: proporcionar um ambiente de desenvolvimento estável; persistir artefatos utilizados e principalmente os que são resultado do desenvolvimento; preservar o histórico de utilização, manutenção e evolução desses artefatos, além de coordenar alterações simultâneas que possam ser efetuadas nos mesmos. Essa perspectiva encontra-se dividida em três sistemas:

- Sistema de Controle de Versões;
- Sistema Controle de Mudanças;
- Gerenciamento da Construção.

Parte da complexidade em se implantar a GCS em uma organização e aplicar satisfatoriamente suas práticas em um projeto reside em atender os variados pontos de vista dos envolvidos. Além disso, outro fator a ser considerado é o fato do relacionamento entre as perspectivas anteriores não ocorrer de modo complementar, mas sobreposto. Desta forma, a Perspectiva de Integração, preocupa-se em prover a interação entre diferentes perspectivas definindo serviços especializados que proporcionarão a integração entre os espaços de trabalho de GCS e o Ambiente de Desenvolvimento de Software (ADS) (Murta, 2006).

Baseado nas perspectivas apresentadas na Figura 2.1, Murta (2006) utiliza uma taxonomia na qual cinco elementos sumarizam as diferentes perspectivas de GCS encontradas na literatura, facilitando sua compreensão e esclarecendo suas aplicações. Esses elementos são denominados: Processo, Sistema de Controle de Versão, Sistema

de Controle de Modificação, Sistema de Gerenciamento de Construção e Integração dos Espaços de Trabalho. Essa taxonomia é utilizada neste trabalho sempre que for necessário fazer referência a algum sistema ou atividade pertencente à essas perspectivas. De acordo com Murta (2006) esses elementos são importantes nos subsistemas em que atuam, e apesar de não abrangerem toda a GCS, provêem funcionalidades que permitem a cada subsistema da Perspectiva de Desenvolvimento atuar nas funções relativas à Perspectiva Gerencial.

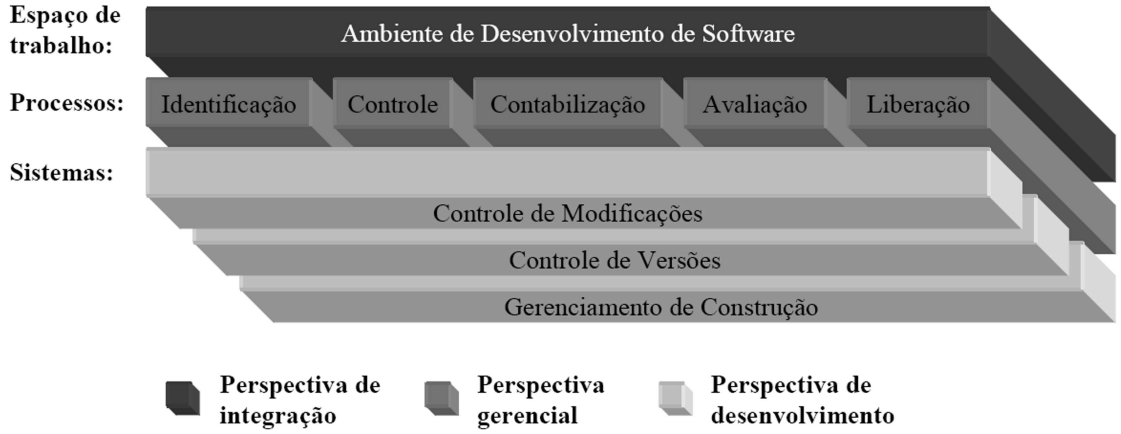

Figura 2.1: Perspectivas da GCS (Murta, 2006)

Este Capítulo está organizado da seguinte maneira: a relação da GCS em Modelos e Normas é comentada na Seção 2.2. A Seção 2.3 apresenta assuntos relacionados a Sistema de Controle de Versões. Nas Seções 2.4 e 2.5, respectivamente, são comentados os sistemas de Controle de Mudanças e o de Gerenciamento de Construções. Comenta-se sobre integração de espaços de trabalho na Seção 2.6, e na Seção 2.7 sobre GCS em LPS. Por último, em 2.8 são feitas considerações finais.

2.2 GCS em Modelos e Normas

Cada modelo de maturidade para desenvolvimento de software possui estratégias próprias para atingir suas metas e todos incluem práticas de GCS como atividade fundamental para apoiá-los. Entre os modelos de processo estão: o CMMI (Ahern et al., 2008) que possui uma estrutura em camadas para a classificação da maturidade de empresas de desenvolvimento de software e estratégias para a melhoria de processo de forma gradual e o MPS.BR (SOFTEX, 2007) que também consiste em um programa para melhoria de processos de software, entretanto é voltado para a realidade de micro, pequenas e médias empresas de software brasileiras.

Existem também diferentes normas internacionais que cuidam do desenvolvimento de software e, conseqüentemente, das práticas de GCS. A norma IEEE Std 1042

(IEEE, 1987) é considerada uma abrangente norma sobre GCS e funciona como um guia para a aplicação da IEEE Std 828 (IEEE, 2005), já a ISO 10007 (ISO, 1995) fornece diretrizes para a utilização de GCS na indústria e define a interface da GCS com as demais áreas de gerência.

Ambos, normas e modelos refletem a experiência dos peritos envolvidos em seu desenvolvimento, e ao utilizá-los pode-se, entre outros propósitos, estabelecer uma relação de confiança para com os produtos de software e também guiar desenvolvedores apoiando-os no processo de produção (Hass, 2003). Devido à importância no contexto deste trabalho, a norma IEEE Std 828 será apresentada em maiores detalhas a seguir.

2.2.1 IEEE std 828

A IEEE std 828 determina requisitos mínimos para o desenvolvimento de um plano de GCS, aplicável durante todo o ciclo de vida de um produto de software. O modelo de plano apresentado é customizável, não se restringindo a nenhum tipo de sistema, podendo inclusive ser utilizado no desenvolvimento de aplicações críticas. O público-alvo dessa norma são os responsáveis pela elaboração do plano de GCS e os envolvidos nas auditorias de GCS. Quanto ao formato do plano, a IEEE 828 determina que ele contenha no mínimo seis seções as quais são denominadas: introdução, gerenciamento, atividades, cronograma, recursos e manutenção do plano.

Na introdução deve constar o propósito do projeto e sua descrição, a identificação dos ICs, a discriminação das ferramentas de hardware e software necessárias à execução do plano, quais atividades do ciclo de vida de software são cobertas pelo plano e também o grau de formalismo que será empregado, e deve constar também todas as restrições de tempo, custos e recursos.

Na seção de gerenciamento determina-se quem é responsável por cada atividade e sua manutenção. Descrevem-se quais unidades organizacionais e indivíduos participam do projeto e quais são seus papéis e atribuições.

A seção de atividades discrimina todas as tarefas necessárias à execução do plano de GCS do software. É a seção mais extensa e encontra-se dividida em sete subseções sendo que cinco dizem respeito ao escopo do projeto: identificação da configuração, controle da configuração, contabilização da situação da configuração, validação e revisão da configuração, gerenciamento de liberação e entrega. As duas subseções restantes são utilizadas apenas quando é preciso identificar artefatos e equipes fora do escopo do sistema em questão.

A seção de cronograma determina a seqüência e a coordenação das atividades registradas no plano. As informações podem ser descritas no formato de datas absolutas ou relativas às demais atividades do projeto. É recomendada a utilização de representações gráficas que auxiliem e esclareçam todas as informações.

Na seção de recursos devem estar registrados todos os equipamentos, ferramentas e técnicas que serão utilizados. Participantes do plano também devem estar registrados nessa seção. É preciso esclarecer detalhes da obtenção de todos os recursos.

É fundamental também garantir a manutenibilidade do plano, que certamente evoluirá durante sua utilização. A incorporação de informações adicionais e customizações para atender projetos específicos também são previstas na norma. Não há uma recomendação específica sobre o formato de divulgação do plano de GCS, desde que as seguintes recomendações sejam seguidas: O plano pode ser disponibilizado individualmente ou anexo a outro documento, entretanto nele deve constar o título "Plano de Gerência de Configuração de Software"; no plano devem constar todas as informações pertinentes, caso seja necessário pode-se referenciar onde outras informações podem ser encontradas. O formato do plano, caso não seja aderente ao formato sugerido, deve estar explicitado na seção de introdução.

É recomendada a criação de uma *baseline* gerencial para controle e evolução dos planos gerenciais, inclusive do plano de GCS, pois esses documentos são também ICs e portanto necessitam ser gerenciados.

A IEEE 1042 (IEEE, 1987) descreve em detalhes a aplicação da disciplina de GCS estando dividida em duas partes, planejamento e implementação. Para apoiar a descrição da etapa de implementação, a título de exemplo quatro planos de GCS são apresentados. Todos foram elaborados segundo a visão da IEEE std 828, fornecendo assim valioso material sobre a aplicação da norma. A IEEE std 828 também é consistente com os processos de GCS definidos pela IEEE/EIA 12207.0 (IEEE, 1996) e pela ISO/IEC 12207 (ISO e IEC, 2003).

2.3 Sistema de Controle de Versões

Sistema de controle de versões (SCV) consiste, basicamente, em um local para armazenamento de artefatos gerados durante o desenvolvimento de sistemas de software (Mason, 2006). A idéia é obter a separação entre artefatos utilizados pelos desenvolvedores também chamados cópias de trabalho, das cópias mestre desses mesmos artefatos armazenadas em um repositório. A principal vantagem é que as cópias armazenadas no repositório não são editadas diretamente. É necessário realizar uma cópia dos artefatos desejados, alterá-los e em seguida enviá-los novamente ao repositório. Cada vez que os artefatos são devolvidos ao repositório, o sistema de controle de versões cria uma nova versão desses artefatos. Assim, cronologicamente todas as versões dos artefatos alterados vão sendo armazenadas no repositório (Louridas, 2006).

Inúmeras soluções, comerciais e também de uso livre estão disponíveis, entretanto, a maior parte dos SCV disponíveis enfocam o armazenamento de código fonte

(Murta, 2006). As próximas Seções apresentam conceitos importantes que permitem um maior entendimento sobre SCV.

2.3.1 Delta

O conceito de deltas leva em consideração que duas versões consecutivas de um mesmo IC possuem entre si, em média, uma similaridade de 98%. O armazenamento das diferenças entre as versões (2%) permite uma otimização do armazenamento (Estublier, 2000). Com a evolução e o aumento da capacidade do hardware, problemas como escassez de memória ou espaço de armazenamento deixaram de influenciar as decisões de projeto, com isso essa abordagem perdeu importância (Estublier et al., 2005).

O termo delta significa a diferença entre duas versões consecutivas de um IC considerado (Leon, 2000). Existem duas abordagens de armazenamento utilizando o conceito de deltas, a técnica de *delta avante* armazena a versão mais antiga do IC e sucessivas diferenças em relação às versões seguintes. A técnica de *delta reverso* armazena a versão mais recente e as diferenças existentes em relação às versões anteriores.

2.3.2 Repositório

É o local "físico" onde são armazenados os ICs e informações a respeito desses ICs (metadados). Alguns SCV utilizam sistemas de arquivos para o armazenamento, outros fazem uso de banco de dados e existem ainda aqueles que utilizam uma combinação das duas abordagens (Mason, 2006).

2.3.3 Espaço de Trabalho, *check in* e *check out*

Espaço de trabalho ou workspace representa a área em que o desenvolvedor pode modificar ICs de modo independente das atividades executadas por outros desenvolvedores (Estublier, 2000). A expressão *check in* significa a cópia de ICs do espaço de trabalho do desenvolvedor para o repositório e *check out* significa a cópia de ICs do repositório para o espaço de trabalho do desenvolvedor (Mason, 2006).

2.3.4 *Trunk*, *branch* e *tag*

O termo *trunk* refere-se ao diretório onde é armazenada a linha principal de desenvolvimento. O termo *branch* é utilizado para referenciar diretórios que contém uma cópia separada da linha principal de desenvolvimento, um *branch* representa uma linha paralela de desenvolvimento. Emprega-se o termo tag para referenciar o diretório reservado para a criação de instantâneos (*snapshots*) de uma versão sendo

considerada. Instantâneos são recursos mnemônicos utilizados para atribuir um rótulo significativo às versões cujas modificações são consideradas de grande impacto ou relevância no contexto do projeto e que se deseja destacar (Mason, 2006).

2.3.5 Políticas de Acesso e *Merge*

SCV que provêem acesso concorrente ao repositório, permitem que vários desenvolvedores trabalhem concomitantemente nos mesmos ICs de um projeto, assim é necessário uma política para controlar as modificações realizadas, de forma que sejam integradas ordenadamente e evitando que um desenvolvedor sobrescreva o trabalho de outro. Para gerenciar essas alterações, existem duas políticas usualmente empregadas, a política pessimista e a otimista.

Na política pessimista não é permitido o paralelismo do desenvolvimento, quando um desenvolvedor realiza um *check out*, todos os ICs relacionados são bloqueados (*lock*) tornando-se indisponíveis para outros desenvolvedores que deverão aguardar pela liberação dos ICs. Com o emprego dessa política evita-se a ocorrência de conflitos de versões, todavia existe a desvantagem de que os demais desenvolvedores devem aguardar pela liberação dos ICs envolvidos, e em caso dessa liberação demorar excessivamente atrasos irão ocorrer no cronograma do projeto. A política otimista permite que os ICs sejam modificados concomitantemente, e trata individualmente eventuais conflitos (Estublier et al., 2005) que são resolvidos utilizando a técnica de *merge* ou junção (Louridas, 2006), na qual as versões conflitantes do IC são contrastadas e o desenvolvedor deve realizar a junção das alterações, validando-as e aglutinando-as ao final um único IC.

2.3.6 *Logs*

SCV utilizam o recurso de criação de *logs* ou registros para documentar as operações realizadas e que alterem o estado do repositório ou dos ICs armazenados. Esse histórico é importante pois permite a qualquer momento o rastreamento e a recuperação de importantes informações sobre o os ICs.

2.4 Sistema de Controle de Modificações

O objetivo principal do Controle de Modificações (CM) é administrar os pedidos de mudança, também chamados de requisições, que ocorrem em um projeto e acompanhar seu ciclo de vida. O CM também deve registrar e relatar o status dos itens de configuração. Um ciclo de vida sucinto para ilustrar uma requisição é composto pelos seguintes passos:

1. Registro da requisição: detalhar o pedido.

2. Análise da requisição: estimativas de custo, impacto, efeitos de propagação da mudança.

3. Julgamento da requisição:

   (a) Se a requisição for aceita, um pedido de mudança é criado, discriminando as alterações solicitadas e listando os ICs envolvidos.

   (b) Se a requisição for rejeitada, uma justificativa deve ser registrada.

4. *Uma requisição aprovada dá início a um novo item de configuração.*

Entre as principais vantagens advindas estão o controle efetivo sobre o escopo do projeto e melhoria da produtividade, pois cada requisição de mudança é analisada de forma coordenada, reduzindo problemas de comunicação entre *stakeholders*. A qualidade pode também ser aprimorada, pois cada modificação deve ser analisada antes de sua implementação, assim seu impacto é analisado previamente.

Existem centenas de ferramentas de apoio ao sistema de controle de mudanças. Uma das mais conhecidas é a ferramenta *open source* Bugzilla (Mozilla Organization, 2008). Outra ferramenta *open source* denominada Trac (Trac, 2008) vem se consolidando como opção estável e produtiva. Ambas ferramentas permitem o rastreamento de mudanças no desenvolvimento de software e utilizam banco de dados relacional e interfaces HTML (W3C, 2008) que garantem um alto desempenho às transações realizadas pelas ferramentas. Entre as funções providas estão: acompanhamento da evolução do projeto, rastreamento de erros e mudanças no código e Wiki para documentação colaborativa. Trac apresenta a vantagem de integração ao sistema de controle de versões Subversion sem a necessidade de ferramentas adicionais.

2.5 Sistema de Gerenciamento de Construções

Durante o desenvolvimento de software diversas tarefas são realizadas tais como, *commits*, *check outs*, geração de artefatos, execução de testes, preparação de distribuições, etc. A construção automatizada de *builds* é uma prática que tem se mostrado eficaz, pois em processos de *build* manuais, o engenheiro de software executa uma série de tarefas e manipula inúmeros artefatos, de forma que essas tarefas estão sujeitas a erros.

Ferramentas de automação de *builds* podem ser configuradas para executar atividades específicas, reduzindo intervenções humanas e a probabilidade de inserção de erros. A construção de uma versão de um sistema – *build* – consiste em reunir um conjunto de ICs fonte que caracterizem uma configuração alvo, e realizar a geração de ICs derivados. Assim, um único usuário pode realizar diversas tarefas em um curto

prazo de tempo. Uma das mais importantes ferramenta de apoio é o Ant. Essa ferramenta possibilita a automatização de tarefas, interpretando instruções contidas em arquivos de configuração (*scripts*) escritos em XML. Esses arquivos são geralmente denominados `build.xml`, apesar de não ser obrigatório esse nome. Entre os avanços pelos quais os sistemas de automação de *builds* passaram estão o acesso aos repositórios dos SCV para obtenção dos ICs fonte, a execução de testes e a execução independente de plataforma (Murta, 2006).

2.6 Integração dos Espaços de Trabalho

Engenheiros de software constantemente são confrontados com complexos problemas que podem ser agravados pela necessidade de integração entre sistemas de controle de versões, controle de mudanças e automação de *builds*. É recomendável que sejam adotadas técnicas que promovam a integração desses sistemas no ambiente de desenvolvimento de software, todavia sem causar impacto à rotina do ambiente de trabalho. O emprego de IDEs como Eclipse e NetBeans pode favorecer essa integração pois são ferramentas que permitem realizar controle de versões, criação de *builds*, execução de testes entre outras tarefas de modo integrado todavia sem o emprego de mecanismos proprietários (Murta, 2006).

2.7 GCS em Linha de Produtos de Software

Linha de Produtos de Software (LPS) consiste em uma família de sistemas de software que possuem um núcleo de funções em comum cuja vantagem é formar um conjunto de recursos reutilizáveis, e funções variáveis que permitem a especialização das aplicações para atender a domínios específicos (Gomaa e Shin, 2007). Atividade de GCS são importantes no controle e evolução de produtos desenvolvidos com base em LPS a fim de evitar sua degradação. Entretanto a despeito dessa importância, Schäfer (1996) afirma que a GCS está muito comprometida com o desenvolvimento e evolução de sistemas tradicionais, provendo pouco suporte ao desenvolvimento de sistemas ditos "não-tradicionais", entre eles as LPS. Krikhaar e Crnkovic (2007) comentam que poucos avanços consideráveis foram feitos no sentido de se reverter esse cenário, chegando mesmo a afirmar que nas pesquisas que tratam de LPS a GCS freqüentemente é negligenciada.

Staples (2004) apresenta alguns problemas característicos do controle de mudanças que são maximizados quando adotados em abordagens de desenvolvimento em LPS.

Complicações na Identificação da Configuração: Controle de mudanças é uma atividade complexa no desenvolvimento de LPS. Em abordagens de desenvolvi-

mento tradicional, a configuração de um produto, usualmente, possui uma estrutura simples. LPS são mais complexas de se controlar, pois tanto variabilidades quanto similaridades evoluem independentemente.

Modificações nas similaridades: Eventuais modificações nas similaridades são propagadas para todos os sistemas gerados após a modificação. Se forem realizadas indiscriminadamente e na ausência de controle de mudanças que permita rastrear qual versão originou determinado produto, a reprodutibilidade desse produto pode ser comprometida.

Diferentes Restrições a cada *Release*: Uma LPS pode possuir diferentes restrições a cada release para atender requisitos específicos de uma liberação. Como os produtos criados compartilham um núcleo de recursos em comum, a liberação de um produto às vezes depende de restrições de liberação contraditórios que foram desenvolvidos para atender outros.

Degradação da Linha de Produto: A degradação de uma LPS pode ocorrer sempre que uma nova funcionalidade é implementada para atender a variabilidade específica de um conjunto de produtos. Os benefícios do desenvolvimento com base em LPS podem deixar de existir se a nova funcionalidade não for suportada pelo núcleo de recursos reusáveis.

2.8 Considerações Finais

Neste Capítulo foram apresentadas abordagens de GCS organizadas do seguinte modo: Processo, Sistema de Controle de Versões, Sistema de Controle de Mudanças, Sistema de Gerenciamento da Construção e Integração dos Espaços de Trabalho. Essa organização, a despeito de não englobar todas as abordagens de GCS apóia os subsistemas em que se pretende utilizá-los (Murta, 2006). O emprego de normas e modelos de processos pode contribuir para guiar o planejamento das atividades de GCS.

A utilização de recursos de automatização de *builds* permite que engenheiros de software indiquem os arquivos alvo em seu projeto, especifiquem as eventuais dependências e como essa versão do programa deve ser construída. As principais vantagens são a minimização do tempo despendido nessas atividades e a redução da probabilidade de inserção de que erros sejam inseridos devido às intervenções manuais (Clark, 2004).

SCV provê controle sobre as diversas versões dos ICs ao longo de seu ciclo de vida. No desenvolvimento e manutenção de sistemas de software mudanças são inevitáveis. Assim, realizar o controle efetivo dessas mudanças, acompanhando todas as requisições e seu impacto sobre o projeto são atividades que colaboram para que o

ICs evoluam de forma disciplinada e em ambiente controlado e rastreado. Isso possibilita que as alterações realizadas nos ICs possam ser revisadas, reproduzidas ou revertidas a um estado anterior (Hass, 2003).

Realizar o controle dos ICs de uma LPS é complexo quando comparado à abordagens de desenvolvimento tradicionais. O cerne dessa complexidade reside no fato de que LPS compartilham um núcleo de funções em comum que freqüentemente é modificado para atender requisitos específicos dos *stakeholders*. O monitoramento constante das modificações de uma LPS pode contribuir para evitar sua degradação (Staples, 2004).

CAPÍTULO
3

Abordagens para Reengenharia de Sistemas Legados

3.1 Considerações Iniciais

Abordagens de produção automatizada de artefatos podem contribuir para a redução de tempo e dos custos empregados no desenvolvimento de software ao transformarem tarefas de desenvolvimento em especificações de alto nível (Krueger, 2000).

Linha de Produtos de Software (LPS) consiste em uma família de sistemas de software que possuem um núcleo de funções em comum, cuja vantagem é formar um conjunto de recursos reusáveis e funções variáveis, que permitem a especialização das aplicações para atender a domínios específicos (Pohl et al., 2005; Gomaa e Shin, 2007). Uma forma de se implementar uma Linha de Produtos de Software e automatizar parte do processo de desenvolvimento é por meio de geradores de aplicações. Geradores aceitam uma especificação, validam-na e geram seus artefatos (Pohl et al., 2005). Uma Linguagem de Modelagem da Aplicação (LMA) é utilizada para representar as abstrações (modelos) das aplicações (Weiss e Lai, 1999) e é uma das formas de documentar essas especificações.

Geradores de aplicações permitem a um desenvolvedor declarar "o que" a ferramenta deve fazer, sem se preocupar em "como" as atividades de automação são realizadas (Smaragdakis e Batory, 2000). Podem ser construídos para gerar não apenas código, mas também casos de testes, diagramas, figuras e documentação (Cleaveland, 2001). Utilizando geradores um desenvolvedor de software pode implementar múltiplos produtos de uma LPS mais facilmente do que utilizando meios tradicionais de

implementação. Entre os benefícios esperados ao se desenvolver geradores de aplicações estão (Franca e Staa, 2002):

Aumento de produtividade: Trechos fixos, referentes a detalhes de implementação, constituem a maior parte de um artefato gerado. A geração automática dessas partes fixas pode proporcionar uma elevação da produtividade da equipe de desenvolvimento.

Redução do tempo para o mercado: O período despendido no desenvolvimento de um novo artefato pode ser reduzido a um tempo equivalente ao tempo desenvolvimento e avaliação da sua especificação.

Prototipação: Alterando-se as especificações que são fornecidas ao gerador novas versões de um artefato são obtidas. Versões experimentais podem ser geradas a custo mais baixo, viabilizando a realização de diversos experimentos para determinar o artefato mais adequado ao usuário.

Qualidade do artefato gerado: A utilização do gerador pressupõe sua correta construção. Por conseguinte é esperada a geração de artefatos de acordo com as especificações. Em caso de problemas, estes estarão associados a problemas na forma de gerá-lo.

A construção de um gerador de aplicação requer a identificação de um domínio e a definição de seu limite. Além disso é preciso identificar as partes variáveis, denominadas variabilidades e as partes invariáveis, comuns a toda aplicação de uma família desse domínio, e que são denominadas similaridades. É preciso ainda definir o método de entrada da especificação, que pode tomar a forma de um diálogo interativo, onde o usuário seleciona as opções que deseja a partir de uma série de menus, ou podem estar na forma de editor gráfico podendo o usuário editar um diagrama. Com isso é possível ao desenvolvedor inserir uma especificação abstrata, e a partir dela o gerador de aplicações realiza, automaticamente, a construção da aplicação (Cleaveland, 2001). Para modificar essa aplicação, é necessário alterar as especificações de entrada e executá-la novamente no gerador de aplicações (Harel, 2002).

Geradores de aplicações podem ser genéricos, contendo grande quantidade de parâmetros, aumentando a variabilidade e abrangência dos geradores e permitindo ampla criação de aplicações. Esses geradores possuem uma estrutura complexa e são considerados difíceis de se implementar. Existem também geradores mais específicos e que são mais fáceis de serem construídos por possuírem uma estrutura simplificada. Esse tipo de gerador possui abrangência menor e caso não seja possível gerar uma determinada aplicação, um outro gerador que atenda ao domínio da aplicação desejada precisa ser construído (Franca e Staa, 2001).

Este Capítulo está organizado da seguinte forma: na Seção 3.2 são apresentadas duas linguagens de padrões de análise importantes no contexto deste trabalho: GRN (Braga et al., 1999) e SiGCli (Pazin, 2004). A Seção 3.3 apresenta o gerador GAwCRe com a funcionalidade original com que foi construído e na Seção 3.4 é comentada uma versão do gerador adaptado para ser utilizado como ferramenta de apoio ao Processo Ágil de Reengenharia com Geradores de Aplicações (Freitas, 2006), também são comentados brevemente a Abordagem de Reengenharia Ágil – ARA (Cagnin, 2005) e o Processo PARFAIT (Cagnin, 2005). Na Seção 3.6 são apresentadas as considerações finais deste Capítulo.

3.2 Linguagens de Padrões de Análise

A engenharia de software se beneficia do uso de padrões pelo que eles representam: o acúmulo de experiências testadas e documentadas de problemas vivenciados por outros profissionais e que descrevem um modo eficiente de como esses problemas foram resolvidos. Um padrão traz em seu cerne a possibilidade de se resolver um problema que se repete em um contexto, podendo ser usado e reusado, no entanto sem recorrer-se à mesma solução mais de uma vez (Alexander, 1977). Padrões são resultantes da experiência adquirida; são identificados e podem ser descritos e documentados (Gamma et al., 1995; Fowler, 2003). A forma de documentação e o conteúdo das informações variam de acordo com a classe de padrões de software que se deseja documentar.

Dentre as principais classificações para padrões de software encontradas na literatura estão: os padrões de projeto (Gamma et al., 1995), de análise (Fowler, 1996), organizacionais e de processo (Coplien, 1996; Ambler e Jeffries, 2002).

Padrões de análise descrevem grupos de conceitos e seus relacionamentos que representam idéias e construções comuns, úteis na modelagem de domínios de negócios. Podem se aplicar a um único domínio ou a vários domínios e apóiam o reúso de idéias durante a fase de análise. Diferente dos padrões de projeto, os de análise não refletem o estado atual da implementação, são estruturas conceituais dos processos de negócios (Fowler, 2005) enfocando as características organizacionais, sociais e econômicas de um sistema, desde que relevantes para a análise de requisitos a aceitação e a usabilidade do sistema final (Geyer-Schulz e Hahsler, 2002).

Padrões de análise contribuem principalmente em duas tarefas no processo de desenvolvimento de software: primeiramente agilizam o desenvolvimento dos modelos de análise abstratos que capturam os principais requisitos de um problema concreto, proporcionando o reúso de modelos de análise, bem como descrições de suas vantagens e limitações. Facilitam a transformação dos modelos de análise para modelos de projeto. Essa transformação a partir do domínio do problema para a resolução

do problema é um processo complicado e consome tempo, assim, padrões de análise auxiliam esse processo sugerindo padrões de projeto e soluções confiáveis para problemas recorrentes (Geyer-Schulz e Hahsler, 2002).

Uma coleção de padrões de uma mesma área de aplicação, estruturados e que se apóiam uns nos outros, e são capazes de transformar requisitos e restrições em uma arquitetura formam uma linguagem de padrões (Schmidt et al., 1996). Cada padrão pode se relacionar a um ou mais padrões que compõem a linguagem e ao menos um padrão deve estar disponível para cada elemento da construção e implementação de um sistema de software não podendo haver "vazios" ou "brancos" (Braga e Masiero, 2002).

A seguir serão apresentadas duas linguagens de padrões de análise importantes no contexto deste trabalho, a Linguagem de Padrões de Gestão de Recursos de Negócios (GRN) (Braga et al., 1999) que é utilizada pelo framework GREN e a Linguagem de Padrões para Sistemas de Gerenciamento de Clínicas de Reabilitação (SiGCli) cujos padrões tiveram por base os padrões da GRN e é utilizada no gerador GAwCRe (Pazin, 2004).

3.2.1 Linguagem de Padrões de Gestão de Recursos de Negócios (GRN)

A Linguagem GRN elaborada por Braga et al. (1999) foi concebida para auxiliar desenvolvedores com pouca experiência a criarem aplicações no domínio de Gestão de Recursos de Negócios. A GRN é composta por quinze padrões, alguns dos quais são extensões ou aplicações de padrões recorrentes na literatura. Entretanto, por ser aplicável ao domínio específico de gestão de recursos de negócios, que é um domínio particular de sistemas de informação, e por possuir valor semântico inerente a uma família de aplicações desse domínio, a GRN encontra-se em um patamar de abstração superior aos padrões de projeto (Braga et al., 1999).

A GRN foi concebida para ser utilizada no desenvolvimento de aplicações que necessitem registrar as seguintes transações:

Transações de aluguel: Considera a utilização temporária de um bem ou serviço;

Comercialização: Transferência de propriedade ou de um bem;

Manutenção: Consiste na restauração ou conservação de um produto.

Os padrões que compõem a GRN são expressos utilizando notação UML (Fowler, 2005), e são agrupados de acordo com o propósito a que se destinam:

- Relacionados à identificação do recurso de negócio;
- Relacionado às transações com os recursos;

- Relacionados aos detalhes das transações.

O grafo ilustrado na Figura 3.1 apresenta uma possível ordem de aplicação dos padrões, os principais padrões são realçados por uma linha reforçada (Braga, 2002). Incluídos no grupo 1 estão os padrões que tratam da identificação e possível qualificação, quantificação e armazenagem dos recursos de negócios, no grupo 2 ficam os padrões que lidam com as transações efetuadas pelo sistema e no grupo 3 os padrões relacionados a detalhes associados à maioria das transações de negócio. Informações adicionais a respeito da GRN podem ser obtidas em Braga (2002).

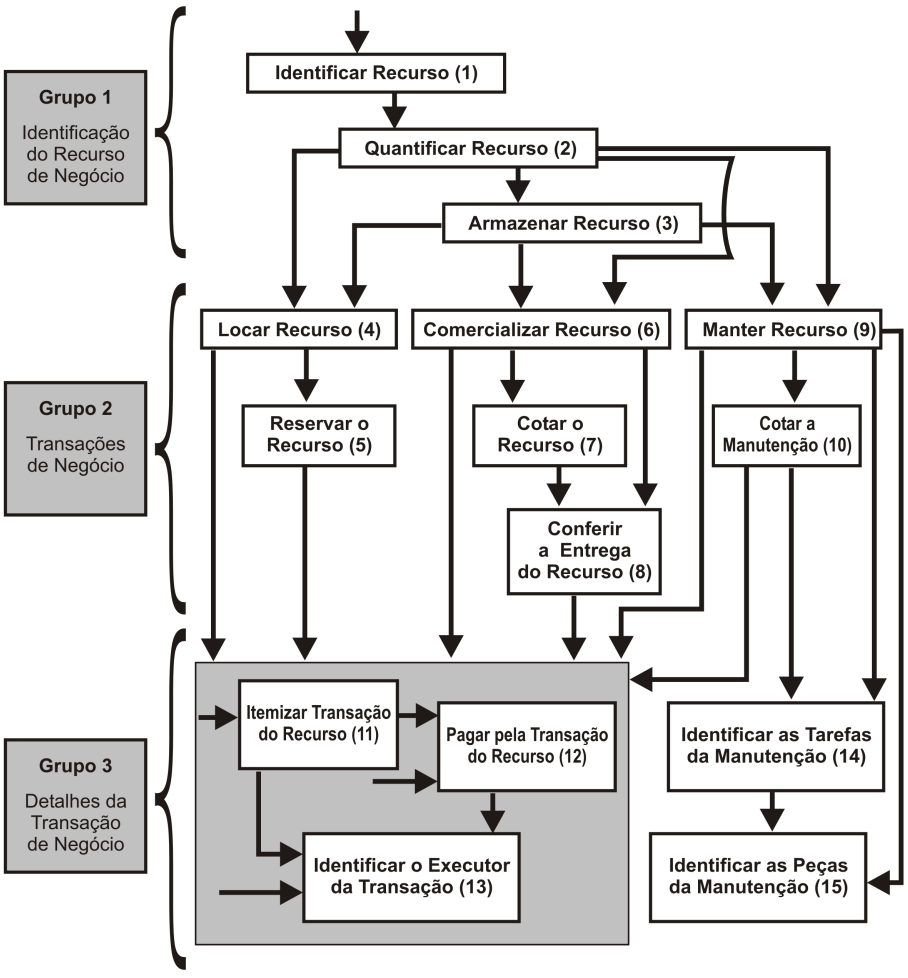

Figura 3.1: Grafo de fluxo de aplicação da GRN (Braga, 2002)

3.2.2 Linguagem de Padrões para Sistemas de Gerenciamento de Clínicas de Reabilitação (SiGCli)

No seu contexto mais genérico, a linguagem de padrões SiGCli se preocupa em obter as informações sobre o paciente e fazer todo o acompanhamento do tratamento através de avaliações contínuas (Pazin, 2004). O domínio da SiGCli inclui: clínicas de fisioterapia, terapia ocupacional e educação física.

Para sua elaboração, foi realizada a engenharia reversa de três sistemas que serviram de base para o estudo do domínio, depois foram elaborados modelos de classes intermediários para representarem cada sistema a fim de se identificar similaridades (Pazin, 2004).

A criação da SiGCli seguiu o processo de construção para linguagens de padrões proposto por Braga (2002). Durante a construção da linguagem observou-se a existência de padrões com características similares a dos padrões da GRN, optou-se então pelo reúso desses padrões sempre que possível. No entanto, nem todas as funções do sistema puderam ser satisfeitas pelos padrões da GRN, sendo necessário realizar adaptações tais como criação de atributos, operações, associações entre classes, entre outras, e um novo padrão necessitou ser criado, pois não havia na GRN nenhum que implementasse sua funcionalidade (Pazin, 2004).

A SiGCli é constituída de 9 padrões reunidos em três grupos. Nem todos os padrões precisam, necessariamente, ser utilizados concomitantemente.

Os padrões do *Grupo 1* devem ser aplicados obrigatoriamente em todos os sistemas desse domínio, trata das informações básicas relacionadas às clínicas, a identificação dos pacientes e dos serviços prestados. O padrões que compõem o *Grupo 2* cuidam do gerenciamento dos atendimentos e os do *Grupo 3* tratam do controle financeiro das clínicas. Os padrões dos dois últimos grupos são opcionais, no sentido em que sua utilização depende do fluxo de aplicação dos padrões que forem utilizados para compor a aplicação sendo instanciada. Os nove padrões que constituem a SiGCli são ilustrados no grafo de fluxo apresentado na Figura 3.2.

As setas existentes entre os padrões mostram que há diversas formas de utilizá-los. Por exemplo:

(i) Quando há necessidade de Realizar Acompanhamento em determinado paciente, os padrões 1, 2, 4, 5, 6 e 7 devem ser utilizados;

(ii) Quando necessitá-se de controle do faturamento, os padrões utilizados são 1, 2, 3 e 9, e assim sucessivamente.

Na próxima Seção será comentado o gerador GAwCRe, que foi elaborado para facilitar as instanciações de aplicações no domínio da linguagem de padrões SiGCli (Pazin, 2004) e define uma família de produtos de software para esse domínio.

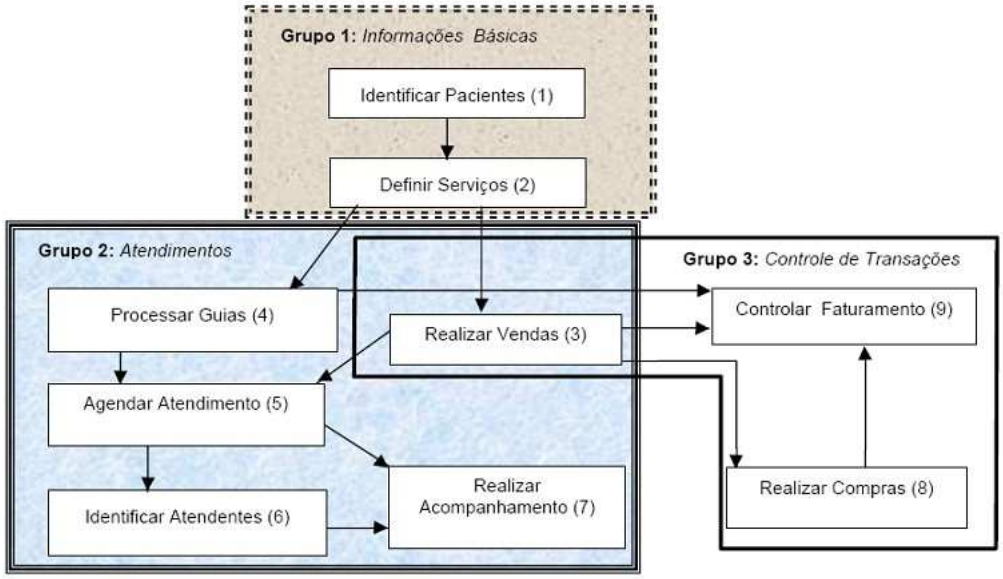

Figura 3.2: Relacionamento entre os padrões da linguagem de padrões de análise SiGCli (Pazin, 2004)

3.3 GAwCRe: Um Gerador de Aplicações para web no domínio de Clínicas de Reabilitação

GAwCRe é um gerador de aplicações para a Web desenvolvido com base em Linha de Produtos de Software, no domínio de clínicas de reabilitação física (Fisioterapia, Terapia Ocupacional e Educação Física). A instanciação do gerador é feita a partir de uma LMA definida com base na linguagem de padrões SiGCli. Para armazenar as informações referentes à linguagem de padrões e à LMA, um meta-modelo em XML foi elaborado contendo um conjunto de *tags* e atributos de *tags*, que são utilizadas para compor os artefatos da aplicação. Na inicialização do GAwCRe, o meta-modelo é lido e com as informações ali contidas, a interface do gerador é definida e os padrões disponíveis para a geração das aplicações são apresentados. O total de aplicações distintas que podem ser geradas pelo GAwCRe, utilizando a SiGCli como foi originalmente criado é de seiscentas e oitenta e oito aplicações. Considerando a flexibilidade da linguagem XML, outras linguagens de padrões podem ser mapeadas no meta-modelo, viabilizando o reúso do gerador em outros domínios (Pazin, 2004).

As aplicações são instanciadas com base em gabaritos de códigos pré-definidos. Os gabaritos possuem partes fixas, comuns a todas as aplicações geradas e também partes customizáveis que são substituídas no momento da criação das aplicações de acordo com os valores definidos no documento XML.

A interface de instanciação das aplicações é construída dinamicamente no mo-

mento em que as definições da LMA são lidas do meta-modelo. As informações da especificação são exibidas na interface de instanciação na forma de *checkbox*, bastando ao desenvolvedor selecionar os padrões e respectivas variantes que deseja implementar. Sempre que um *checkbox* é selecionado, as informações são armazenadas na base de dados do gerador, possibilitando posteriormente, a recuperação e a alteração da especificação LMA para cada aplicação gerada.

A seleção dos padrões ocorre com a utilização de botões de navegação. Alguns padrões são optativos, devem ser escolhidos ou não de acordo com os requisitos da aplicação que está sendo instanciada, entretanto outros padrões são obrigatórios e o avanço para o próximo padrão só é possível após a seleção do padrão corrente. Após a seleção do último padrão, os artefatos podem ser gerados a partir do menu do instanciador e são então automaticamente gravados em um diretório criado pelo gerador. Para armazenamento de dados do gerador e das aplicações criadas a partir dele foi utilizado o SGBD Oracle (Oracle, 2008).

Na Figura 3.3 são ilustradas, sucintamente, as etapas de criação de uma aplicação utilizando o gerador. Tipos de artefatos produzidos pelo GAwCRe:

Scripts de criação do banco de dados: O menu *Gerar SQL* aciona o módulo gerador de scripts SQL que gera a estrutura do banco de dados Oracle (Oracle, 2008) para a aplicação.

Classes Java: O menu *Gerar Código* aciona o módulo gerador das classes Java (*beans*) que têm as regras de negócios da aplicação.

Interface Web da aplicação: O menu *Gerar Interface* aciona o módulo gerador de interfaces JavaServer Pages (JSP) (Sun Microsystems, Inc., 2008) que cria as interfaces Web em que o usuário final interage com a aplicação e que são visualizadas no navegador (*browser*).

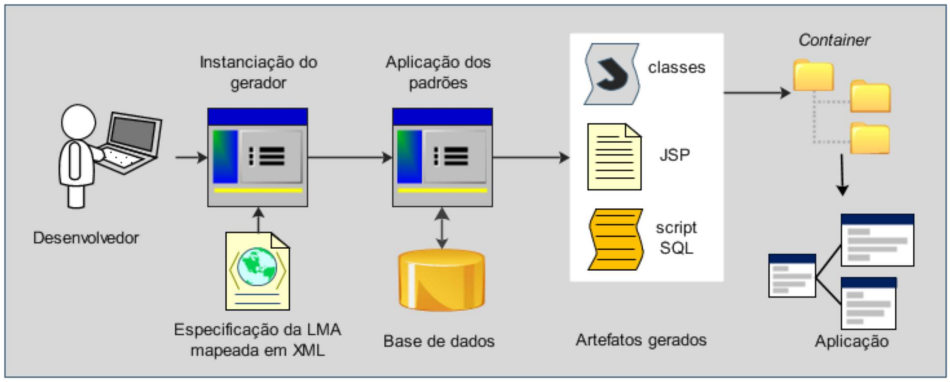

Figura 3.3: Visão geral da criação de uma aplicação no domínio da SiGCli

O GAwCRe utiliza também algumas bibliotecas escritas em JavaScript (Flanagan, 2002) para todas as aplicações criadas. Elas são responsáveis pelo funcionamento do menu das aplicações geradas e devido ao uso comum não são geradas para cada aplicação (Pazin, 2004).

As aplicações geradas pelo GAwCRe são sistemas baseados na Web e são desenvolvidas utilizando o conceito de arquitetura em três camadas. As solicitações e interações dos usuários são feitas na camada de apresentação. As páginas são geradas pela camada de aplicação e exibidas por meio de um navegador, quando necessário, esta se comunica com a camada de persistência que é responsável pelo armazenamento e pela recuperação das informações do sistema. A Figura 3.4 ilustra a arquitetura das aplicações geradas pelo GAwCRe. Após a geração da aplicação, deve-se disponibiliza-las em um servidor Web configurado para interpretar código Java.

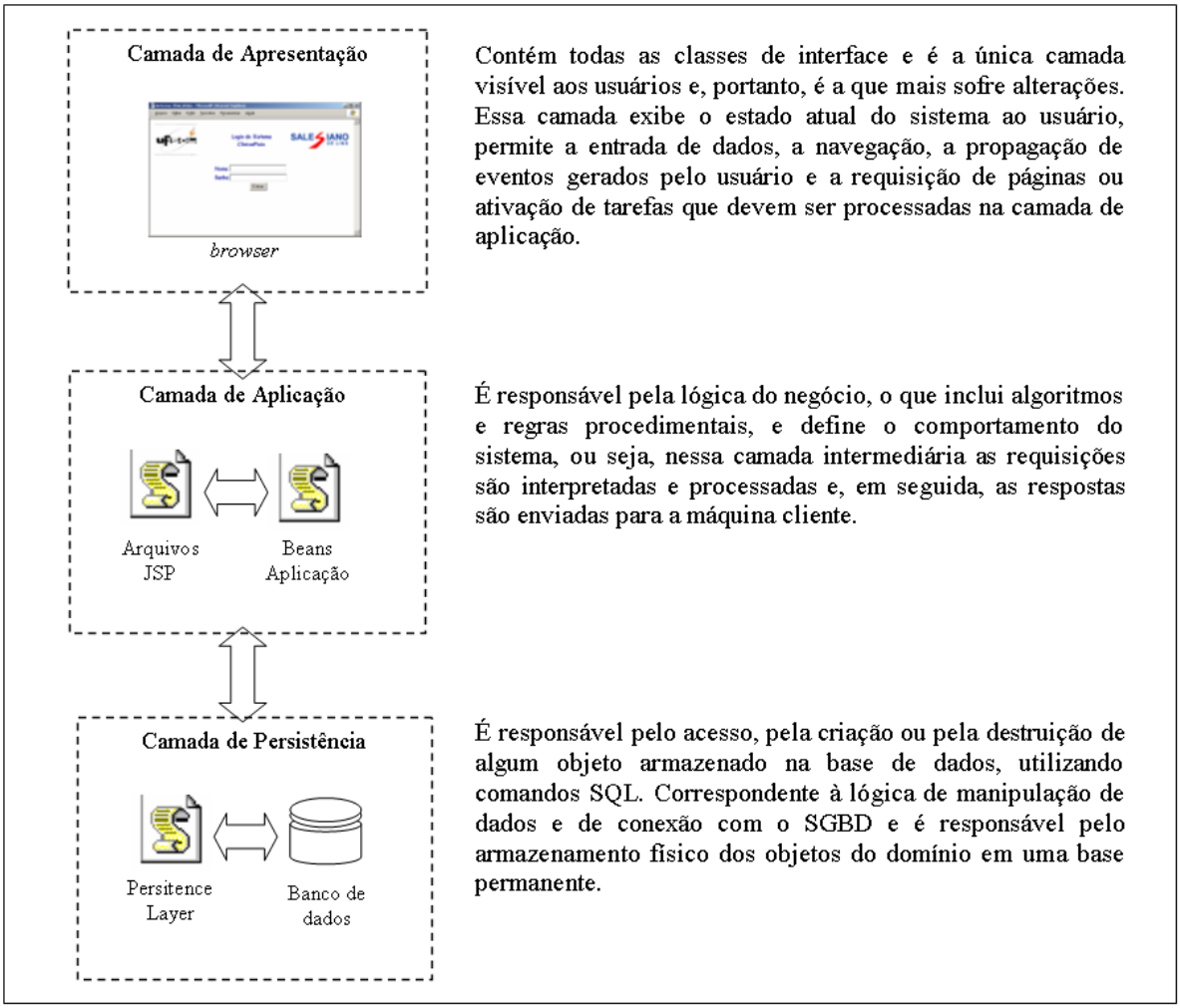

Figura 3.4: As aplicações geradas pelo GAwCRe são desenvolvidas utilizando o conceito de arquitetura em três camadas (Pazin, 2004)

3.3.1 Substituição do SGBD Oracle pelo SGBD MySQL

GAwCRe originalmente utilizava o SGBD Oracle (Oracle, 2008) e após um trabalho de iniciação cientifica realizado por Rizzo (2005) foi substituído pelo SGBD MySQL. Esse SGBD foi escolhido considerando seu uso livre e por ser amplamente difundido. O GAwCRe utilizava um leitor de código XML (*parser*) proprietário da Oracle (Oracle, 2008) que também foi substituído e em seu lugar passou-se a utilizar o *parser* Xerces (Apache Xerces, 2007) de uso livre.

Na próxima Seção apresenta-se o Arcabouço de Reengenharia Ágil (ARA) (Cagnin, 2005) e a utilização do GAwCRe juntamente com esse arcabouço.

3.4 ARA: Arcabouço de Reengenharia Ágil

Arcabouço de Reengenharia Ágil (ARA) (Cagnin, 2005) apóia a reengenharia de sistemas legados procedimentais para sistemas orientados a objetos. ARA pretende ser uma abordagem completa, apoiando o processo de reengenharia em todas as etapas, desde a recuperação do modelo conceitual até a substituição do sistema legado pela nova implementação. Para que isto seja possível, ARA conta com abordagens de reúso em vários níveis de abstração e com a utilização de práticas ágeis, modelos de processos, ferramentas de apóio, testes e versionamento de código, como apresentado na Figura 3.5. A utilização desses recursos visa à redução do tempo, dos custos e alguns dos riscos de reengenharia identificados por Rosenberg (1996).

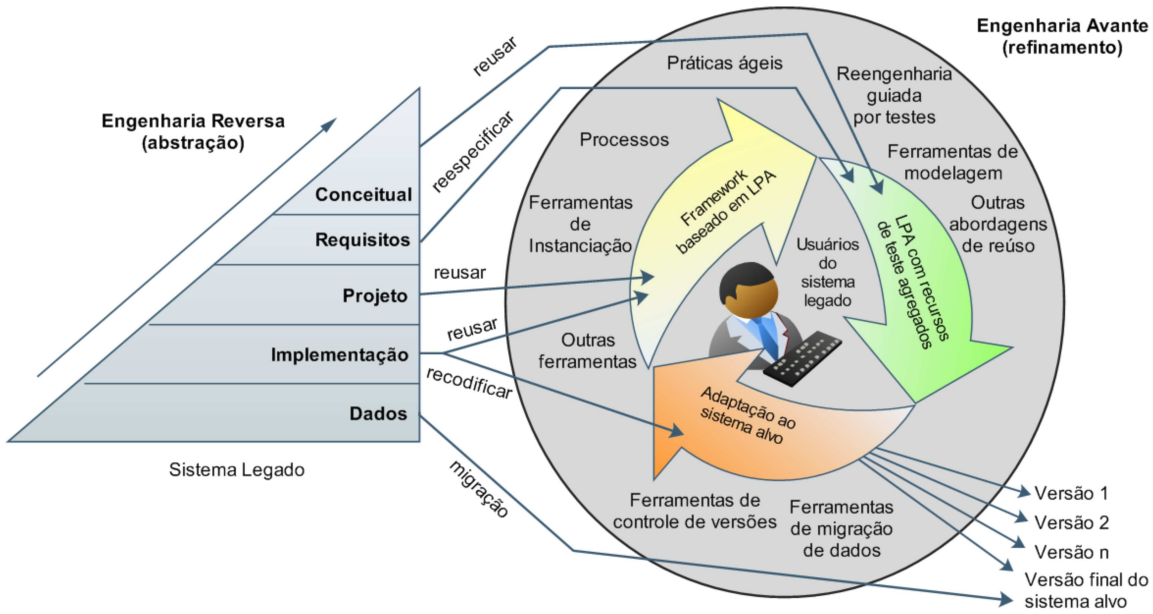

Figura 3.5: Abordagem ARA (Cagnin, 2005)

ARA obtém os conceitos, o entendimento e a análise do sistema legado apoiado por uma linguagem de padrões de análise e também por recursos de teste funcional, sendo que o sistema legado deve pertencer ao mesmo domínio da linguagem de padrões utilizada. Já o projeto e a implementação do sistema são, em grande parte, reutilizados do framework, e todas aquelas funções que o framework não engloba e que são específicas do legado ou são identificadas posteriormente pelos usuários são implementadas diretamente no sistema alvo (Cagnin, 2005).

A prioridade, para cada requisito a ser submetido à reengenharia, pode ser definida pelos usuários, assim como suas validações podem ser realizadas em versões parciais do sistema graças à abordagem incremental e iterativa. O objetivo dessas validações é verificar se as funções implementadas são equivalentes às do sistema legado, para tal, são reutilizados os mesmos casos de teste utilizados anteriormente durante a etapa de engenharia reversa (Cagnin, 2005).

Os benefícios no uso da abordagem ARA são baseados nos mesmos benefícios do uso de métodos ágeis, pois, é adaptativo e tem participação dos usuários e clientes, é incremental e utiliza testes desde o inicio da reengenharia, incentiva a programação em pares, garante a propriedade coletiva do código por meio de controle de versão de todos os artefatos produzidos durante a reengenharia, incentiva a jornada de trabalho de no máximo quarenta horas semanais e garante a prática de metáforas, além disso, através da linguagem de padrões de análise e do uso de classes herdadas do framework, aumenta a confiabilidade do sistema alvo, uma vez que essas classes já foram testadas.

Um processo ágil de reengenharia baseado em framework e com atividades de VV&T denominado PARFAIT, foi proposto por (Cagnin, 2005) para ser utilizado juntamente com ARA. Esse processo será apresentado a seguir.

3.4.1 PARFAIT: Processo Ágil de Reengenharia baseado em Framework no domínio de sistemas de Informação com técnicas de Validação, Verificação e Teste

O processo PARFAIT foi criado para ser associado ao arcabouço ARA com o objetivo de apoiá-lo em atividades de reengenharia, mais especificamente na migração de sistemas legados procedimentais para sistemas baseados no paradigma orientado a objetos (Cagnin, 2005).

O processo é dividido em quatro fases, sendo que cada fase é composta por um conjunto próprio de atividades. A fases do PARFAIT têm o mesmo nome que as do *Rational Unified Process* (RUP) (Kruchten, 2000); Concepção, Elaboração, Construção e Transição; entretanto possuem objetivos específicos para o contexto de reengenharia e agrupam atividades com objetivos em comum o que facilita a documentação do

processo.

A fase de Concepção contém atividades que incluem a identificação do escopo e do domínio do sistema legado em relação ao framework considerado devendo ser levados em consideração os riscos associados. A fase de Elaboração possui atividades que promovem o entendimento do sistema legado com base na execução de casos de teste. Na fase de Construção, uma versão operacional do sistema deve ser liberada rapidamente, a partir da instanciação do framework, sendo adaptada a cada iteração até que os requisitos do sistema alvo sejam equivalentes ao do sistema legado. Por último, a fase de Transição contempla a preparação do sistema alvo e sua implantação no ambiente definitivo. Ao final de uma atividade, artefatos são produzidos ou atualizados, e ao final de cada fase do processo, um marco de referência é elaborado para permitir que o responsável pela reengenharia verifique o estado atual do projeto e defina se a continuação da reengenharia é viável ou não. Como o modelo do processo é incremental e iterativo, a cada iteração é possível a retomada de qualquer atividade a fim de se refinar os artefatos produzidos. Todos os artefatos obtidos durante a reengenharia são colocados sob controle de versões (Cagnin, 2005). Na Figura 3.6 a visão geral do processo PARFAIT é apresentada com suas fases, atividades e com a indicação de quais passos são obrigatórios e quais não são.

PARFAIT atende diversas práticas do XP (Beck, 2000), entretanto algumas dessas práticas tais como: refatoração constante, padrão de codificação e projeto simples, só podem ser utilizadas, caso o responsável da reengenharia verifique a necessidade e providencie meios para aplicá-las. PARFAIT também atende à Modelagem Ágil (Ambler e Jeffries, 2002), que busca a modelagem e a documentação do sistema legado de maneira efetiva e ágil, facilitando o entendimento e comunicação entre desenvolvedores (Cagnin, 2005).

O uso de framework como apoio computacional permite que, uma vez obtidos os requisitos do sistema, uma versão inicial e operacional do sistema alvo seja criada e disponibilizada. Essa versão evolui de forma incremental durante o processo até atingir a versão final. A participação dos usuários é incentivada desde o início, por isso, a cada iteração, as versões produzidas são baseadas na prioridade que os usuários atribuem aos requisitos em função da importância que eles possuem para o negócio. Além disso, todos os artefatos produzidos podem ser constantemente avaliados pelos usuários, permitindo um controle maior sobre a qualidade do sistema resultante, evitando que subsistemas inadequados sejam selecionados e descartando a recuperação de informações inúteis (Rosenberg, 1996).

Durante o processo, componentes do sistema legado não são mantidos em conjunto com componentes do sistema alvo. Todas as versões do sistema alvo são disponibilizadas para os usuários para fins de testes de aceitação, portanto o sistema legado continua operacional até a conclusão do processo de reengenharia. Essa

estratégia permite que as versões parciais do sistema alvo sejam apresentadas aos usuários concomitantemente ao sistema legado a fim de validá-las, procurando assim garantir a equivalência funcional até que a versão final seja obtida.

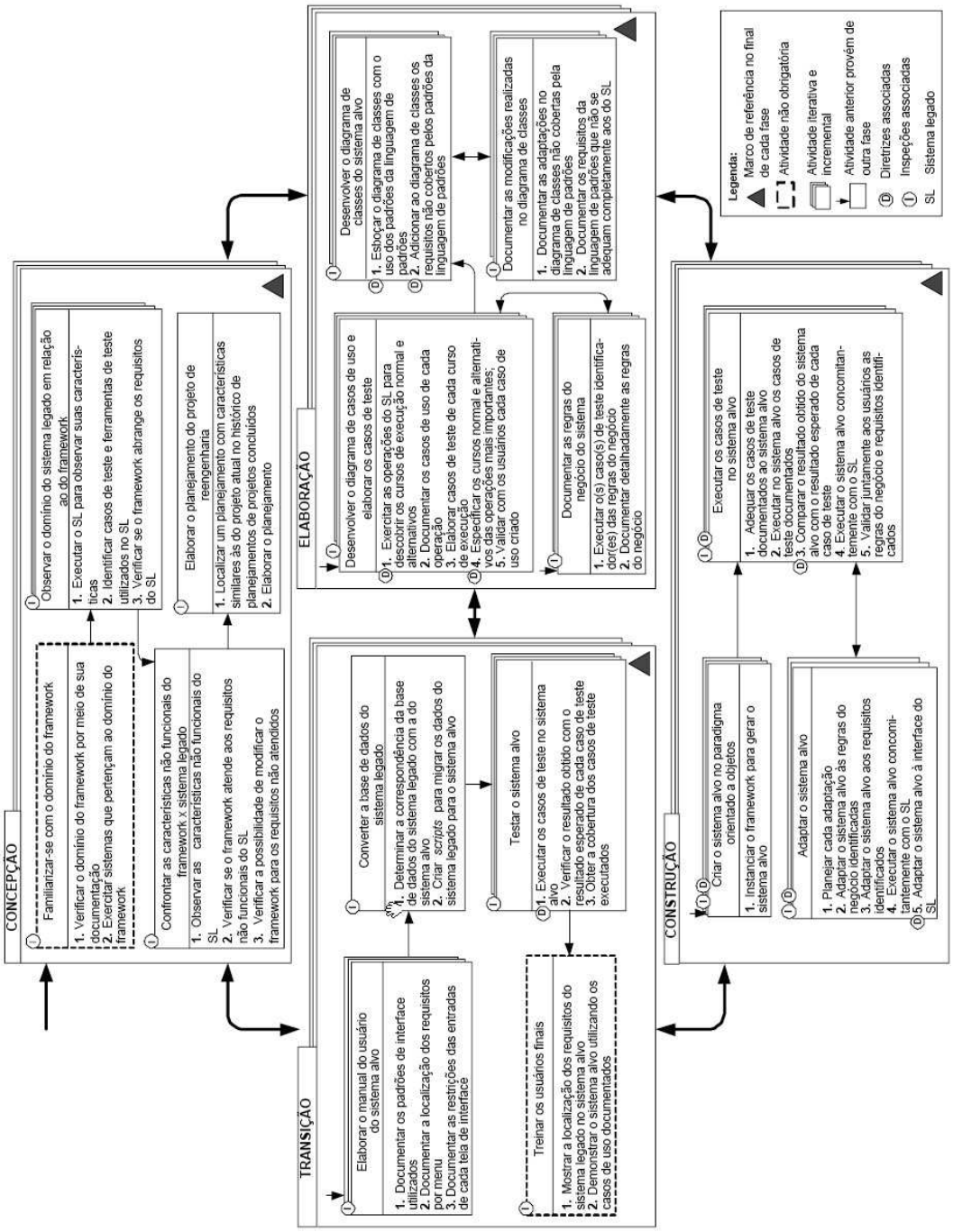

Figura 3.6: Visão geral do processo PARFAIT

Caso ocorram alterações de funcionalidade no sistema legado durante o processo de reengenharia, as atividades pertinentes podem ser retomadas e redefinidas para que os novos requisitos sejam incorporados ao sistema alvo, sempre de acordo com as prioridades estabelecidas. Para utilização efetiva do processo PARFAIT, a título de pré-requisito, faz-se necessária a familiaridade com os seguintes temas (Cagnin, 2005):

- Conhecimento do Paradigma Orientado a Objetos;
- Conhecimento do domínio do framework;
- Conhecimento da linguagem de padrões;
- Conhecimento da linguagem de programação em que o framework foi construído;
- Conhecimento do mecanismo de persistência utilizado para o armazenamento de dados do framework e dos sistemas legados;
- Conhecimento da linguagem UML (*Unified Modeling Language*);
- Conhecimento dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite.

Quanto mais proficiente nesses pré-requisitos for o engenheiro de software, maiores as chances de redução de alguns dos riscos de reengenharia citados por Rosenberg (1996), por exemplo: "alto custo da reengenharia" e "ausência de conhecimento ou de experiência do pessoal envolvido".

A documentação de cada atividade do PARFAIT é composta pelos itens apresentados na Tabela 3.1.

Para avaliar o processo PARFAIT e o arcabouço ARA, o framework GREN (Braga, 2002) foi utilizado. Esse framework instancia sistemas em linguagem de programação Smalltalk (Sharp, 1997), pertencentes ao domínio de gestão de recursos de negócio e foi construído a partir da linguagem de padrões GRN.

3.5 Uso do Gerador de Aplicações GAwCRe como Ferramenta de Apoio no ARA

Por um processo *ad hoc* (Freitas, 2006), foi proposta a utilização do gerador GAwCRe como ferramenta para a geração automatizada de artefatos no Arcabouço de Reengenharia Ágil ARA (Cagnin, 2005). Devido à dificuldade em se encontrar sistemas exemplo pertencentes ao domínio do gerador GAwCRe, foi realizada a reengenharia de sistemas pertencentes a domínios conexos ao originalmente implementado. Assim, foi necessária a expansão do GAwCRe para atender essa expansão.

Tabela 3.1: Itens que compõem a documentação do processo PARFAIT

	ITEM
1)	Papel de quem deve executar a atividade.
2)	Quais são os artefatos necessários para se iniciar a atividade.
3)	Quais são os artefatos de saída resultantes da atividade.
4)	Detalhamento dos passos para apoiar a execução da atividade.
5)	Ferramentas que podem ser usadas como apoio computacional para facilitar a execução da atividade.
6)	Guia de uso das ferramentas que não sejam familiares ao(s) responsável(eis) pela reengenharia.
7)	Inspeções no formato *checklist* para garantir que os artefatos produzidos são aqueles esperados.
8)	Gabaritos em que são baseados os artefatos de saída.
9)	Algumas atividades e passos são executados utilizando técnicas específicas e são apoiados por diretrizes a eles associadas.

O framework denominado GREN foi substituído pelo gerador de aplicações GAwCRe, no arcabouço ARA, para que fosse verificada a viabilidade do uso de geradores de aplicação em uma abordagem de reengenharia ágil, apoiando a produção automatizada de artefatos (Freitas, 2006), que é uma atividade considerada complexa quando se utilizam frameworks (Eisenecker e Czarnecki, 2000).

A Seção seguinte comenta dificuldades e vantagens com o uso do GAwCRe juntamente com o ARA.

3.5.1 Processo Ágil de Reengenharia utilizando Gerador de Aplicações

O Processo Ágil de Reengenharia com Geradores de Aplicações (Freitas, 2006) foi elaborado com base no processo PARFAIT. O objetivo desse processo é apoiar a reengenharia de sistemas legados, associado ao arcabouço ARA, e utilizando de geradores de aplicação como ferramentas de apoio à geração automática de código, em substituição ao uso de frameworks. Ambos, frameworks e geradores de aplicação são formas de se implementar uma arquitetura de LPS, possuindo, portanto pontos em comum que tornam possível a substituição, como por exemplo:

- Tanto frameworks quanto geradores promovem o reúso de projeto e de código (Johnson, 1997);
- As tarefas de configuração dos *hot-spots* em um framework equivalem ao fornecimento das especificações do artefato a ser gerado (Franca e Staa, 2001).

- Frozen-spots equivalem às partes fixas dos artefatos provenientes de geradores e as partes variáveis correspondem aos *hot-spots* (Franca e Staa, 2001).

No arcabouço ARA, Figura 3.7, o símbolo * indica as adaptações em relação à sua versão original, ou seja:

- A linguagem de padrões GRN (descrita na Seção 3.2.1) é substituída pela linguagem de padrões SiGCli (descrita na Seção 3.2.2);
- O gerador GAwCRe, com o banco de dados relacional MySQL, é utilizado como ferramenta de apoio para geração automática de artefatos em substituição ao framework GREN;
- Não é realizado controle de versões.

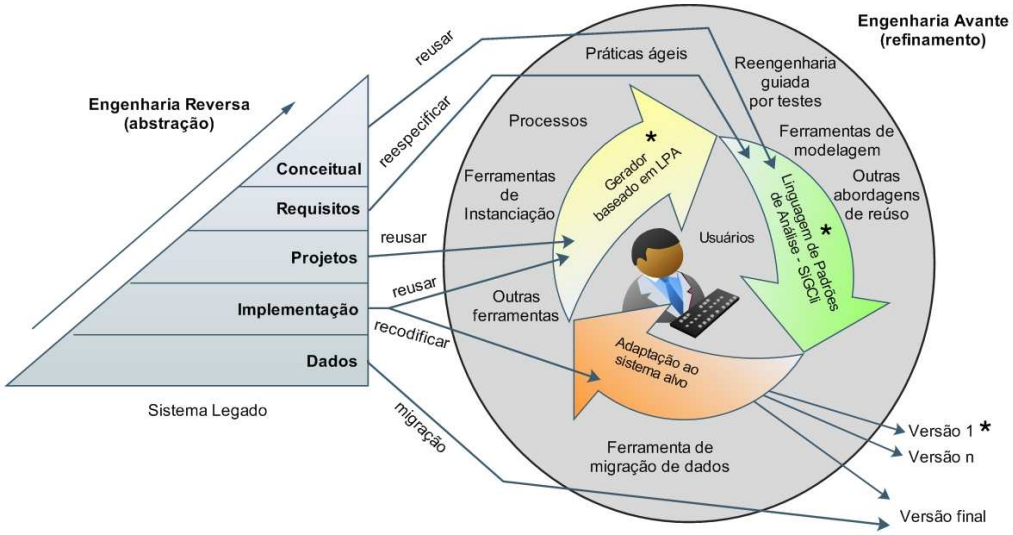

Figura 3.7: Arcabouço de Reengenharia Ágil apoiado por gerador de aplicações

Como comentado anteriormente, sistemas pertencentes a domínios conexos ao da linguagem de padrões SiGCli foram utilizados no processo de reengenharia. Um sistema de atendimento à clínica de psicologia, denominado *Psychologist*, foi utilizado para averiguar a possibilidade da utilização do GAwCRe juntamente com ARA. As adaptações realizadas no processo ágil de reengenharia com o uso de geradores de aplicações são comentados a seguir.

No Processo Ágil de Reengenharia com Geradores de Aplicações (Freitas, 2006), as quatro fases que constituem o processo PARFAIT: Concepção, Elaboração, Construção e Transição, foram mantidas, entretanto, não foram utilizadas todas as atividades como originalmente planejadas, devido às diferenças de utilização entre geradores e

frameworks. Geradores não permitem a instanciação de sistemas, somente a utilização das classes construídas para o seu funcionamento. Dessa forma, diferentemente de frameworks, o gerador de aplicações possibilita o uso das classes com o mesmo nome e atributos que constam do repositório do gerador. Assim, algumas atividades sofreram adaptações e duas novas atividades foram criadas como mostrado na Tabela 3.2.

Tabela 3.2: Atividades previstas no Processo Ágil de Reengenharia com Geradores de Aplicações (Freitas, 2006)

FASE	**ITEM**	
Concepção	1. Familiarizar-se com o domínio do gerador	A
Concepção	2. Buscar sistemas legados	C
Concepção	3. Verificar domínio dos sistemas legados	C
Concepção	4. Observar o domínio do sistema legado em relação ao gerador	A
Elaboração	5. Desenvolver o diagrama de casos de uso e elaborar os casos de testes	U
Construção	6. Criar o sistema alvo no paradigma orientado a objetos	A
Construção	7. Executar os casos de teste no sistema alvo	U
Construção	8. Adaptar o sistema alvo	A
Transição	9. Desenvolver o diagrama de classes do sistema alvo.	A
Transição	10.Converter banco de dados.	U
Transição	11. Testar o sistema alvo.	U
Legenda: A→ Adaptada C→ Criada U→ Utilizada		

A atividade três é necessária devido à restrição de domínio existente no gerador. A atividade dois é adicionada pois o gerador é usado para reengenharia de sistemas legados e não para o desenvolvimento de novos sistemas.

A finalidade da fase de Concepção é permitir que o engenheiro de software entenda o domínio da ferramenta de geração dos artefatos e o domínio do sistema legado e avalie se a reengenharia é possível. A busca por sistemas legados foi uma necessidade para a realização do estudo de caso, e na prática talvez possa ser desconsiderada.

Durante a fase de Elaboração, todas as funções do sistema legado devem ser estudadas para permitir que o engenheiro reflita sobre a ordem em que as funções serão implementadas no sistema alvo. O número de iterações e a quantidade de requisitos que serão atendidos em cada iteração, na ausência de usuários reais, são uma decisão do engenheiro. Para apoiar essas decisões devem ser elaborados diagramas de caso de uso e devem ser criados casos de teste.

Na fase de Construção o sistema alvo é criado, e deve evoluir até que as funcionalidades do sistema gerado atendam aos requisitos do sistema legado.

A última fase prevista é a de Transição, nela os últimos testes são realizados e a documentação finalizada, os dados são migrados e o engenheiro deve assegurar-se de que o sistema alvo está pronto para ser disponibilizado aos usuários.

A partir dos requisitos do sistema legado, as funções priorizadas para a implementação são instanciadas no gerador e uma versão 0 do sistema alvo é criada. As funções da versão 0 são comparadas às funções priorizadas e se existirem discrepâncias o gerador sofre adaptação, especificamente, o documento XML recebe *tags* e atributos de *tags* para corrigir as diferenças.

Uma nova versão do sistema alvo é criada, e também deverá ser testada para que se confirme ou não se atende aos requisitos priorizados do legado. Esse ciclo se repete até que o sistema final atenda a todos os requisitos do legado. Nessas atividades o gerador passa por sucessivas adaptações e por fim, a última versão do gerador atende ao subdomínio do sistema legado. Essa versão final do gerador pode ser utilizada como versão 0 em futuras migrações de outros sistemas legados desde que eles pertençam a esse mesmo subdomínio, a qual essa versão está preparada para atender. Dessa forma o desenvolvedor pode contar com várias versões de geradores que atendem a subdomínios de clínicas médicas, com a linguagem SiGCli sempre sendo utilizadas e apenas os requisitos específicos do legado são alterados (Freitas, 2006).

Novos sistemas legados que pertençam ao subdomínio de clínicas médicas, podem usar a versão original do GAwCRe como versão zero junto ao Processo Ágil de Reengenharia com Geradores de Aplicações e se apresentarem requisitos específicos esses devem ser tratados diretamente na XML.

Esse processo realizado por Freitas (2006) não permite que um desenvolvedor utilize alguma versão produzida que não seja a versão 0 ou a final do gerador. Dessa forma, se houver um sistema pertencente ao domínio de clínicas médicas que seja necessário passar por um processo de reengenharia, ou seus requisitos podem ser atendidos pelos padrões da SiGCli (versão 0 do GAwCRe) ou pelas modificações realizadas anteriormente. Nenhum controle de versões foi utilizado para que versões intermediárias pudessem ser utilizadas.

3.6 Considerações Finais

Neste Capítulo foram apresentados conceitos sobre LPS e geração automatizada de artefatos de software utilizando geradores de aplicações. O entendimento de algumas características de LPS e também de linguagem de padrões de software, frameworks e geradores de aplicação são importantes nesse contexto por estarem relacionados ao tema desta dissertação: SiGCli, GRN, GAwCRe, PARFAIT e ARA.

Atividades de manutenção de software são importantes, pois é cada vez maior o

número de sistemas em operação que necessitam ser atualizados. Reengenharia é uma forma de manutenção que permite que novas tecnologias ou requisitos sejam incorporados a sistemas em operação ao invés de descartá-los. As adaptações realizadas no GAwCRe, foram necessárias para permitir a reengenharia de sistemas legados pertencentes a domínios conexos àquele originalmente utilizado no gerador.

Para se obter sistemas alvo cujos requisitos equivalem aos do sistema legado, foi proposto por Freitas (2006) que as adaptações necessárias para se obter essa equivalência sejam realizadas no mapeamento XML e não no código fonte do sistema alvo. Essa abordagem apresenta a vantagem de manter a agilidade do processo ao mesmo tempo em que se evitam intervenções manuais no código fonte, contribuindo para minimizar a inserção de erros. Entretanto a ausência de um controle de versões para controlar as diferentes versões do gerador GAwCRe pode representar um problema. Atividades de GCS no desenvolvimento com base em LPS são consideradas fundamentais, pois apóiam a preservação, tanto das ferramentas de geração de código, quanto dos produtos gerados (Schäfer, 1996; Hass, 2003; Staples, 2004; Yu e Ramaswamy, 2006).

CAPÍTULO
4

Gerência de Configuração de Software para Apoiar a Evolução do Gerador GAwCRe

4.1 Considerações Iniciais

As atividades de Gerência de Configuração de Software (GCS) são importantes para o desenvolvimento e evolução de sistemas de software, pois, controlam as modificações de forma que a consistência e a integridade do software sejam asseguradas. No desenvolvimento de Linhas de Produtos de Software (LPS), o suporte provido pela GCS é ainda mais importante, considerando que diversos produtos podem ser construídos com similaridades e variabilidades pertinentes (Thao et al., 2008). Em uma LPS, é desejável que o conjunto de recursos que implementa a funcionalidade, bem como o que implementa a variabilidade, estejam sob algum tipo de gerenciamento que permita a evolução controlada da LPS e também dos produtos desenvolvidos, evitando que alterações inapropriadas aconteçam (Staples, 2004).

Devido aos problemas abordados durante a reengenharia de aplicações utilizando o gerador GAwCRe (Capítulo 3), constatou-se a necessidade de um processo de Controle de Versões (CV) para auxiliar a evolução desse gerador e dos artefatos produzidos. Sem um processo de CV, quando o gerador é alterado a fim de possibilitar a geração de aplicações pertencentes a domínios conexos, todo o conhecimento prévio relacionado ao gerador é perdido. De modo que, normalmente, não é mais possível gerar artefatos, exatamente com a mesma funcionalidade produzida pela versão an-

terior do gerador. Assim, caso seja necessário gerar artefatos de acordo com aqueles produzidos inicialmente pelo GAwCRe, todas as alterações realizadas devem ser manualmente desfeitas. Isso resulta em sobrecarga de trabalho e aumenta as chances de se introduzir problemas.

Um processo de GCS para apoiar a manutenção e evolução do gerador GAwCRe e de seus artefatos é proposto. Esse processo de GCS tem os seguintes objetivos:

- Estabelecer a configuração de cada versão do gerador;
- Gerenciar o desenvolvimento e evolução do gerador;
- Gerenciar o desenvolvimento e a manutenção de artefatos criados por meio do gerador.

Os passos e atividades necessários para a definição do processo são descritos neste Capítulo que está organizado do seguinte modo: na Seção 4.2 comenta-se sobre o processo de GCS proposto; na Seção 4.3 são descritas as principais atividades e tarefas que foram realizadas para a elaboração e implantação de um plano de GCS para apoiar esse processo e, por fim, a Seção 4.5 apresenta as considerações finais.

4.2 Processo de Gerência de Configuração

A definição do processo de GCS visa a atender duas necessidades: i) apoiar a evolução e manutenção do GAwCRe e, por conseguinte da SiGLi, prevenindo a ocorrência de problemas no controle de mudanças tais como os apresentados por Staples (2004) e que foram percebidos na utilização do gerador quando utilizado com o arcabouço ARA; ii) dar apoio à geração de artefatos provenientes do processo de reengenharia, evitando alguns dos riscos apontados por Rosenberg (1996).

Para apoiar o processo de GCS proposto, um plano de GCS foi elaborado, com base na norma IEEE 828-2005. Conforme apresentado no Capítulo 2, essa norma define um plano de GCS composto por seis seções: *Introdução*, *Planejamento da Gerência*, *Atividades de GCS*, *Cronograma*, *Recursos* e *Manutenção do Plano de GCS*. Cada seção apresenta tarefas específicas a serem realizadas. Algumas dessas tarefas foram adaptadas e/ou simplificadas, com o propósito de facilitar a "integração" do processo de GCS definido para a utilização do GAwCRe em substituição a um framework no arcabouço ARA. Adaptações são previstas pela norma IEEE 828-2005 e, desse modo, a agilidade do processo PARFAIT, conforme descrito no Capítulo 3, é preservada. Na Tabela 4.1 estão listadas as seções do plano de GCS definido e as atividades que o compõe.

Tabela 4.1: Seções do Plano de GSC proposto (IEEE, 2005)

SEÇÃO	TAREFA	DESCRIÇÃO
Introdução	–	Descrever o propósito do plano de GCS, o escopo de sua utilização, abreviaturas, glossário e referências.
Planejamento da GCS	–	Identificar papéis, responsabilidades e quais atividades devem constar no plano.
Atividade	Identificação da Configuração	Identificar os artefatos que necessitam ser gerenciados.
Atividade	Controle da Configuração	Processo de controle de modificações dos ICs.
Atividade	Relatório do *Status* da Configuração	Reportar aos envolvidos o andamento do projeto e eventuais modificações nos ICs.
Organograma	–	*A ser definido.*
Recursos	–	Descrever as ferramentas de apoio e demais recursos necessários à execução do plano.
Manutenção do Plano de GCS	–	Definir como o plano deve ser reexaminado e atualizado para se adequar à evolução do projeto.

4.3 Elaboração do Plano de Gerência de Configuração de Software

O escopo deste trabalho enfatiza as seções de *Planejamento da Gerência*, *Atividades de GCS* e *Recursos*, detalhando suas características e funcionalidades. Nas próximas Subseções, comenta-se a realização do plano de GCS que foi elaborado durante a realização deste projeto. Detalhes sobre as Seções de *Introdução*, *Cronograma*, e *Manutenção do Plano de GCS*, são disponibilizados no Apêndice A.

4.3.1 Planejamento da Gerência

Durante essa atividade um documento foi elaborado, contendo as seis seções previstas na utilização da norma IEEE 828-2005 e também as atividades consideradas mais coerentes às necessidades do processo. Durante o desenvolvimento do projeto, o plano foi atualizado e, sempre que necessário, algum refinamento ou ajuste de alguma informação ou tarefa, que não haviam sido definidos inicialmente ou necessitaram ser reexaminadas, foram realizados. Foram também identificados e nomeados os

papéis dos participantes do projeto. Esses papéis são classificados de acordo com as respectivas tarefas executadas. Definiu-se também as tarefas que compõem a seção *Atividades*, estabelecendo o papel de quem pode executá-las e os recursos necessários. As tarefas *Auditoria* e *Gerência de Liberação* e *Entrega*, não foram consideradas neste trabalho. As tarefas de *Identificação da Configuração*, *Controle da Configuração* e *Relatório do Status da Configuração* são apresentadas na próxima Subseção.

4.3.2 Atividades de GCS

Nesta Subseção são apresentadas três atividades requeridas para o gerenciamento da configuração de sistemas de software. Ressaltando que tais atividades foram realizadas no gerador de aplicações GAwCRe.

Identificação da Configuração

Uma das decisões mais difíceis, durante a implementação do processo proposto, foi a escolha de quais artefatos deveriam ser controlados. A identificação da configuração depende integralmente da escolha dos Itens de Configuração (IC), sendo um ponto chave para a GCS de linha de produtos (Schäfer, 1996; Staples, 2004). Para estabelecer os referenciais (*baselines*) para utilização do gerador bem como apoiar sua manutenção foi preciso "recuperar sua história": obter todas as versões existentes do gerador.

No início dos trabalhos, apenas os arquivos fonte (classes Java, *scripts* SQL e o mapeamento XML) utilizados por Freitas (2006) estavam disponíveis. Utilizando o IDE Eclipse uma estrutura de projeto foi criada para satisfazer a disposição dos arquivos fonte pertencentes ao GAwCRe. Após a realização dessa atividade, algumas atividades de manutenção corretiva foram realizadas a fim de que o gerador fosse instanciado.

Visando a familiarização com a utilização do GAwCRe, algumas tarefas foram realizadas, e são descritas a seguir:

- O gerador foi empregado para a criação de algumas aplicações hipotéticas pertencentes ao domínio da SiGLi. Durante a instanciação dessas aplicações, avaliou-se a usabilidade da interface gráfica com o usuário (GUI), o fluxo de aplicação dos padrões da linguagem SiGLi e os tipos e quantidade de artefatos gerados.
- Leitura do código-fonte do gerador e dos artefatos relacionados às aplicações produzidas.
- O *plugin* para o IDE Eclipse denominado PMD 5.0 (PMD, 2008) foi usado com o propósito de analisar o código fonte do gerador em busca de problemas como, por exemplo, variáveis e métodos não utilizados e código duplicado.

Após essas três atividades foram detectados os problemas apresentados na Tabela 4.2:

Tabela 4.2: Problemas que impediam a utilização efetiva do gerador

O código fonte não segue uma convenção de codificação;
A lógica de programação utilizada para implementar alguns métodos é complexa de ser assimilada, dificultando a manutenção desses métodos;
Faltam instruções para a instalação e operação do gerador;
O registro e a validação das manutenções anteriores é superficial ou, na maioria dos casos, inexistente;
A instalação dos artefatos gerados não é automatizada e faltam instruções para realizá-la;
Há a presença de bad smells (Fowler et al., 1999) como, por exemplo, código duplicado, classes e métodos extensos, etc.

Como o GAwCRe já havia passado pelas manutenções comentadas no Capítulo 3, foi proposta a recuperação de suas versões anteriores com o propósito de se compreender adequadamente a evolução do gerador. Dessa forma, a definição dos ICs não se baseou somente na última versão que estava disponível (Freitas, 2006) quando este trabalho teve início (Borges et al., 2008), mas englobou todas as versões, possibilitando a recuperação de informações relevantes sobre as alterações que, anteriormente, não foram suficientemente documentadas. As versões do gerador foram então recuperadas, organizadas em ordem cronológica e armazenadas, temporariamente, em subdiretórios criados usando a seguinte convenção: `(nome_do_autor_ou_mantenedor-ano_de_desenvolvimento)`, como pode ser observado na Figura 4.1. Esses diretórios foram criados utilizando somente recursos do sistema operacional, ou seja, nenhuma ferramenta de apoio foi utilizada.

A próxima tarefa foi definir um modelo de biblioteca controlada (*library*) para o armazenamento definitivo dos ICs. No primeiro modelo proposto, como pode ser observado na Figura 4.2, os ICs foram divididos em três grupos de acordo com sua função em relação à arquitetura do gerador:

- Grupo *Producao*: reúne o conjunto de classes responsáveis pela geração das aplicações e da interface de instanciação do GAwCRe. Nesse grupo também se encontra o pacote pl, que contém as classes que implementam o padrão *Persistent Layer* (Yoder et al., 1998).

- Grupo *LingPadrao*: contém os arquivos que possuem os meta-modelos mapeados em linguagem XML, tanto o arquivo com o mapeamento original da SiGCli quanto "variantes" desse arquivo que foram adaptados para domínios conexos.

- Grupo *Produto*: engloba todos os artefatos instanciados a partir do gerador, visto que cada aplicação tem sua própria linha de desenvolvimento os artefatos são facilmente relacionados ao gerador usado na sua produção (Yu e Ramaswamy, 2006).

Três repositórios foram criados, um para cada grupo (Clark, 2004), todos seguindo o modelo trunk, tag, branch como uma forma de padronização. Após a criação dos

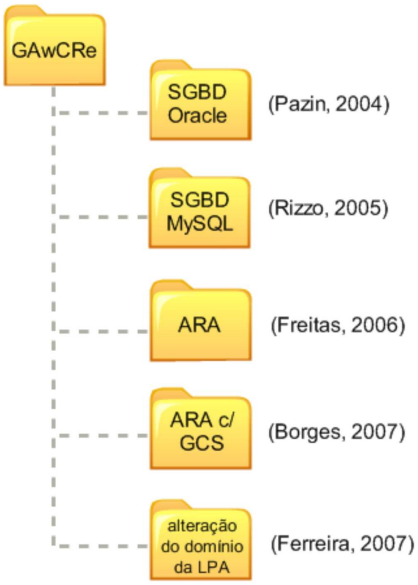

Figura 4.1: Estrutura de diretórios onde foram, provisoriamente, armazenadas as várias versões do gerador GAwCRe

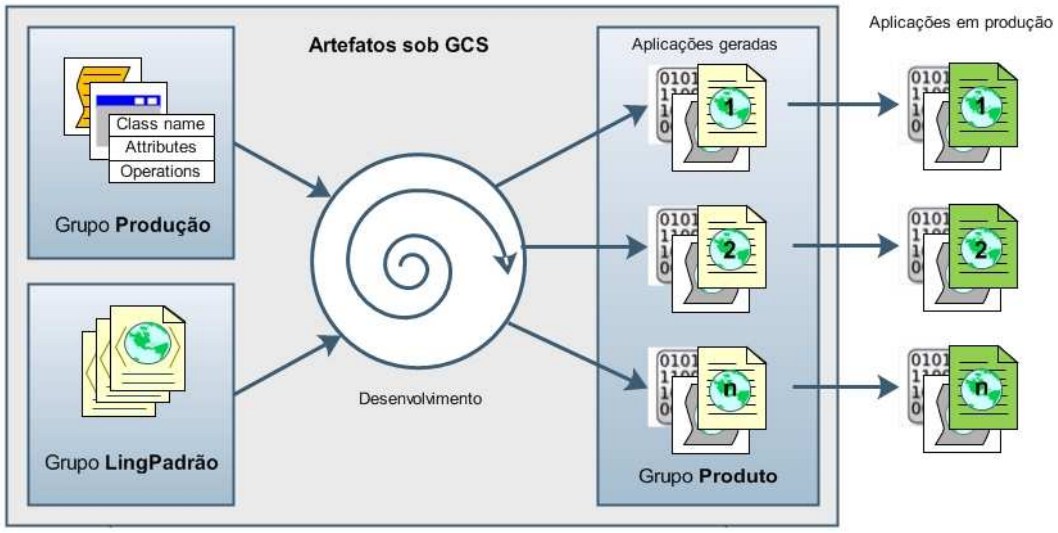

Figura 4.2: Primeiro modelo de agrupamento de ICs proposto; adaptado de (Yu e Ramaswamy, 2006)

repositórios, foi feita a importação dos ICs para os seus respectivos diretórios.

- Repositório do Grupo *Producao*:
  - *trunk*: A versão zero do gerador foi armazenada;
  - *branch*: Foram criados subdiretórios, um para cada versão do gerador;
  - *tag*: não utilizada.
- Repositório do Grupo *LingPadrao*:
  - *trunk*: Foi armazenado o mapeamento original da Linguagem de Padrões de Análise (LPA) SiGLi em formato de arquivo XML;
  - *branch*: Foram criados subdiretórios, um para cada versão da LPA SiGLi;
  - *tag*: não utilizada.
- Repositório do Grupo *Producao*:
  - *trunk*: Reservado para armazenar as aplicações geradas pelo GAwCRe. Cada aplicação é armazenada em um diretório próprio identificado pelo nome da aplicação;
  - *branch*: não utilizada;
  - *tag*: não utilizada.

A criação e manipulação de três repositórios, apesar de viável para alguns tipos de projetos, não foi satisfatória nesse caso. Com a utilização desse formato, tarefas como a criação de *workspaces* e a realização de *checkouts* demandam diversas intervenções manuais para a manipulação dos ICs, aumentando as chances de introdução de problemas e o tempo de execução dessas atividades. Além disso, a construção automatizada de configurações torna-se mais complexa, com a necessidade de criação de *scripts* de automação que manipulem dois ou mais repositórios concomitantemente (Mason, 2006).

O modelo proposto, Figura 4.2, foi então reestruturado. O conteúdo dos grupos *Producao* e *LingPadrao* foi reunido em um único grupo denominado *Gerador* que corresponde ao repositório onde estão armazenados todos os ICs do gerador GAwCRe. A disposição do grupo *Produto* não foi alterada. Essa nova disposição pode ser observada na Figura 4.3.

No repositório correspondente ao Grupo *Produto*, as aplicações são armazenas mantendo a mesma estrutura com que são geradas pelo GAwCRe. Cada uma é armazenada em seu próprio diretório e esses diretórios são armazenados no diretório principal (*trunk*) (Clark, 2004), conforme ilustrado na Figura 4.4.

Para armazenar os ICs pertencentes ao repositório do Grupo *Gerador*, os subdiretórios foram organizados de acordo com o modelo de disposição denominado *Jakarta Directory Layout* (Apache Jakarta, 2008), que foi elaborado com base em sugestões de desenvolvedores que colaboram nos projetos mantidos pela fundação Apache. O *layout* sugerido colabora para a organização dos diversos tipos de artefatos produzidos por diferentes equipes durante todo o ciclo de vida do sistema sendo considerado.

A adoção desse modelo trouxe vários benefícios, entre eles facilitou a manutenibilidade e a compreensão do repositório (Clark, 2004) e permitiu a criação de um

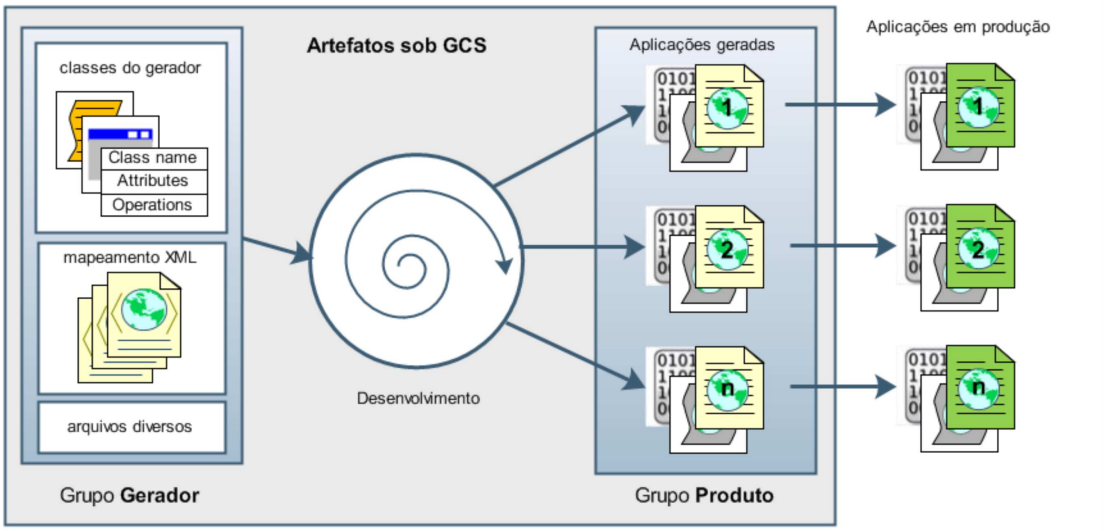

Figura 4.3: Disposição atual dos artefatos sob versionamento; adaptado de (Yu e Ramaswamy, 2006)

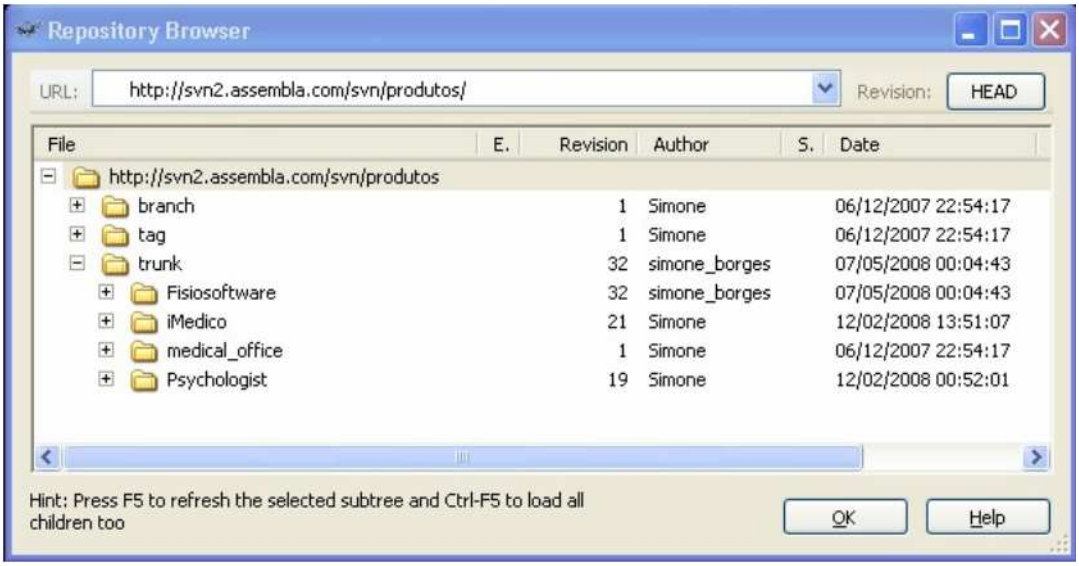

Figura 4.4: Conteúdo do repositório do Grupo *Produto*

vocabulário comum a todos os envolvidos. Além disso, por ser o mesmo modelo de diretórios empregado pelo Tomcat (Apache Tomcat, 2008), é "compatível" com as aplicações geradas pelo GAwCRe (Pazin, 2004).

Na Figura 4.5 é possível observar o conteúdo do repositório do Grupo *Gerador*. Na raiz do path está a *url* do repositório, hospedado no servidor Assembla, descrito na Seção 4.3.3. No segundo nível, encontram-se os diretórios *trunk*, *tag* e *branch*. Os ICs mantidos no subdiretório `...\trunk\GAwCRe` correspondem à primeira versão do gerador (Pazin, 2004). Os detalhes da função de cada subdiretório e as regras e estratégias de utilização estão documentadas no Plano de GCS no Apêndice A.

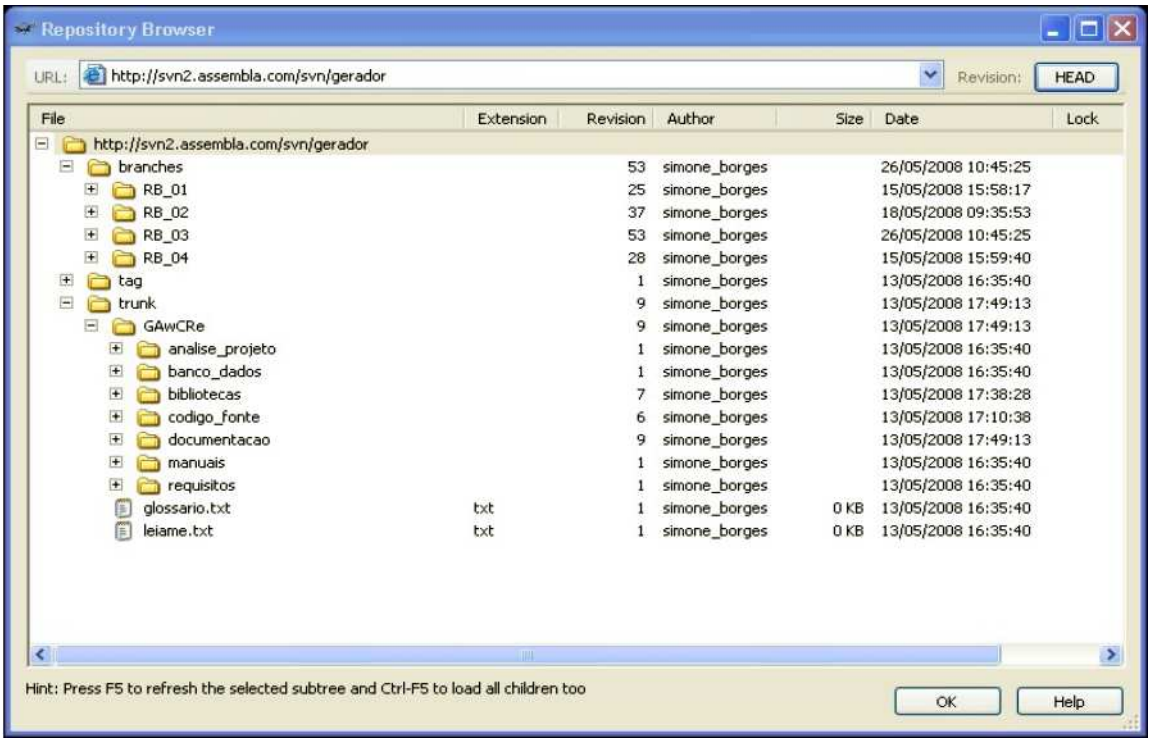

Figura 4.5: Conteúdo do repositório do Grupo *Gerador*

A denominação dos diretórios contidos em `...\branches` segue a convenção: `RB_número_da_versão` onde `RB` é a abreviatura de *Release Branch*. Cada subdiretório do diretório branches contém uma versão do gerador GAwCRe, desenvolvida a partir da "versão inicial" de Pazin (2004):

- `RB_01`: armazena a versão do gerador desenvolvida por Rizzo (2005);
- `RB_02`: armazena a versão do GAwCRe adaptada por Freitas (2006);
- `RB_03` e `RB_04`: contém as duas últimas versões do gerador que passaram por atividades de manutenção, realizadas respectivamente por Borges (2007) e Ferreira (2007).

O diretório *tags* é reservado para a criação de instantâneos (*snapshots*) de uma versão sendo considerada. Esses instantâneos são recursos mnemônicos úteis, pois permitem o registro de modificações ocorridas no projeto em um determinado momento no tempo. À medida que o projeto avança, compreender qual a funcionalidade exata de uma versão, como por exemplo, `RB_03.01.11`, pode requerer uma ou mais consultas à documentação e ao repositório, entretanto uma versão rotulada como `correção_bug_conexao_BD` pode ser mais esclarecedora. Assim, quando uma versão é copiada para esse diretório recebe um rótulo (*tag*) significativo, que auxilia o entendimento dos propósitos da modificação ocorrida. A utilização de *tags* não é obrigatória e elas não devem ser utilizadas para a criação de novas linhas de desenvolvimento, *tags* apenas documentam a realização de eventos como a criação de *releases*, *builds*, ou a realização de modificações significativas (Mason, 2006).

Controle de Configuração

O objetivo da tarefa *Controle da Configuração* é manter o registro de todas as solicitações e realizações de modificações nos ICs armazenados nos grupos *Gerador* e *Produto*. Um processo de requisições de mudanças, ilustrado na Figura 4.6, deve ser seguido antes que qualquer alteração seja realizada em um ICs (IEEE, 2005). Todas as solicitações de mudanças devem ser registradas na ferramenta de Controle de Mudanças. Tais solicitações são analisadas e, caso o pedido seja negado, uma justificativa para a rejeição deve ser registrada. Uma solicitação aceita leva a um subprocesso de implementação. Concluída a implementação, essa deve ser analisada. Caso satisfaça os requisitos, todos os ICs alterados são persistidos no repositório e um relato da modificação realizada deve ser registrado junto ao pedido de modificação que deu origem ao processo. O fluxo dessas atividades é ilustrado na Figura 4.6.

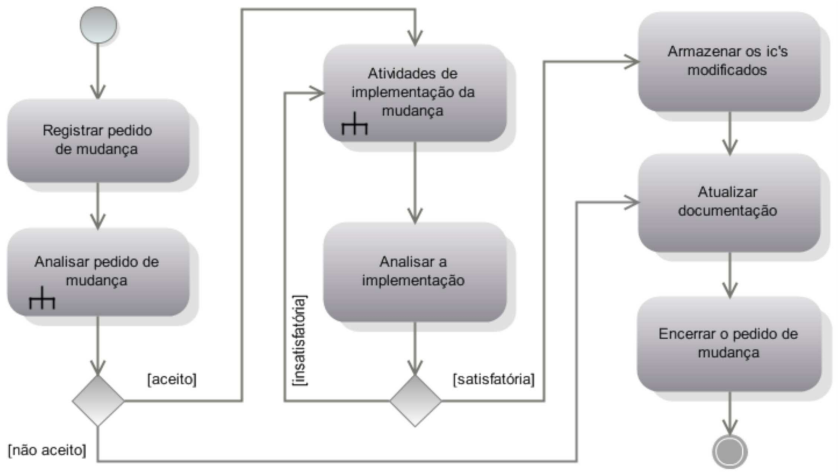

Figura 4.6: Atividades de Requisição de Modificações (IEEE, 2005)

A Figura 4.7 ilustra a abordagem de GCS baseada em evolução (Yu e Ramaswamy, 2006) proposta para controlar essas modificações. Solicitações de mudança que são aceitas e, normalmente, envolvem alterações em alguma versão do gerador, implicam em uma atualização dos sistemas já existentes que foram produzidos pela versão considerada. Modificações nas aplicações armazenadas no repositório *Produto* também são controladas e, quando essas aplicações são atualizadas, se necessário *releases* podem ser entregues.

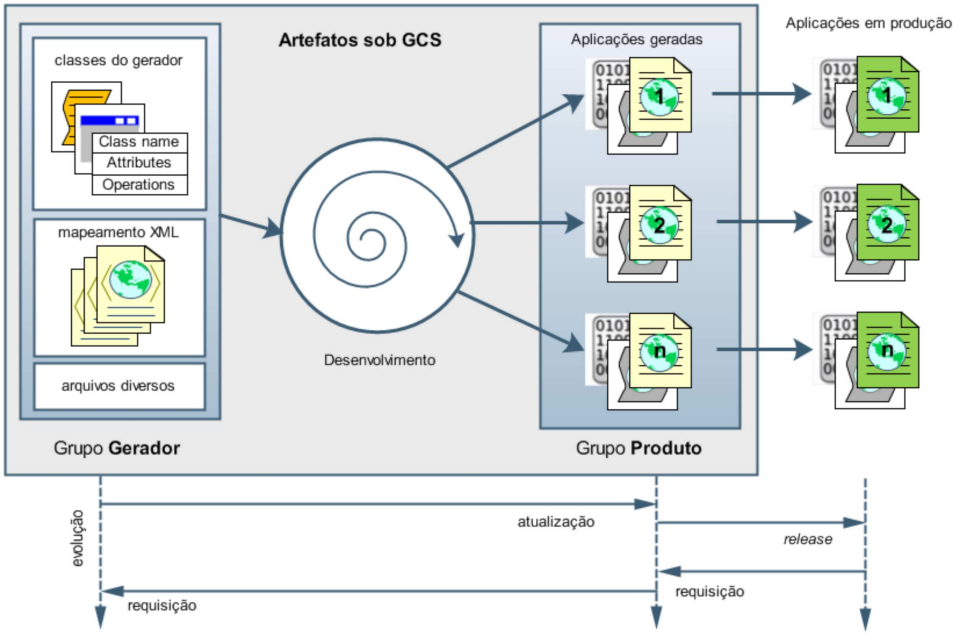

Figura 4.7: Modelo de GCS baseado em evolução para controlar modificações no Gerador GAwCRe e nas aplicações geradas; adaptado de Yu e Ramaswamy (2006)

Requisições de modificação também podem ser feitas a partir da necessidade de manutenção em aplicações em produção. O responsável decide e justifica a melhor abordagem para atender essa requisição: (i) pode-se alterar a aplicação (sistema alvo) (Cagnin, 2005) ou (ii) pode-se gerar novamente a aplicação utilizando uma das versões do gerador GAwCRe (Freitas, 2006).

Relatório de *Status* da Configuração

Essa atividade tem por objetivo manter os participantes do projeto informados sobre alterações realizadas. A ferramenta de controle de mudanças utilizada neste trabalho provê recursos de documentação on-line (Louridas, 2006) em formato wiki (Haley, 2007) e permite a visualização das solicitações de mudanças, dos registros das alterações e a visualização do conteúdo do repositório sendo considerado. Dessa maneira todos os envolvidos no projeto podem obter informações sobre as modificações efe-

tuadas e o motivo de sua realização, bem como o(s) autor(es) e quais itens foram modificados.

A realização da atividade de Identificação da Configuração contribuiu para a recuperação de todas as versões existentes do gerador. Os ICs foram identificados e armazenados sob o controle da GCS, possibilitando o registro e o rastreamento das modificações realizadas. Esses registros podem ser consultados e informações sobre o andamento do projeto tornam-se disponíveis para todos os participantes. Todavia, o uso eficaz dessas informações e sua manutenção dependem da integração dos espaços de trabalho de GCS (Murta, 2006) em que são utilizadas. Para prover essa integração, recursos precisam ser disponibilizados, tais como infra-estrutura e ferramentas de apoio. Os recursos que apóiam a execução dessas atividades são comentados a seguir.

4.3.3 Recursos

Um servidor local foi configurado no laboratório GDMS/DC-UFSCar para disponibilizar os serviços necessários à realização das atividades previstas no plano de GCS definido. Entretanto, devido às restrições de segurança relacionadas à rede do departamento, foi inviável disponibilizar o servidor para conexões externas. Assim, decidiu-se pelo uso de um servidor de projetos gratuito e on-line com suporte às ferramentas de apoio selecionadas para prover os sistemas necessários à realização dessas atividades. Entre os pesquisados, o Assembla (*Accelerating Software Development*) (Assembla, 2008) foi considerado o mais adequado, pois provê todas as funções para o controle de versões e controle de mudanças e também permite a criação de *backup* local dos projetos hospedados. Desse modo, caso eventualmente o portal Assembla apresente problemas e torne-se indisponível para a hospedagem do gerador GAwCRe e seus artefatos, todo o conteúdo pode ser migrado para outro servidor, que disponibilize esses sistemas, sem prejuízos ao projeto.

Os passos para a criação do perfil de usuário e do espaço do projeto (chamado *my space* no portal) estão disponíveis na documentação online do portal (Assembla, 2008).

Ferramentas de Apoio

GCS é altamente dependente da utilização de ferramentas, pois elas automatizam tarefas repetitivas e "não-criativas" (Hass, 2003). Algumas ferramentas possibilitam a automação de todo o processo de GCS, porém, são sistemas de software com alto preço no mercado. Existem diversas soluções livres para automatização de atividades de GCS. A utilização dessas ferramentas em conjunto pode satisfazer plenamente às necessidades de um projeto GCS, provendo a integração dos ambientes de trabalho e

minimizando a complexidade apresentada por certas atividades de GCS, tais como o controle de versões e o controle de mudanças (IEEE, 1987).

A definição das ferramentas de apoio utilizadas neste trabalho foi realizada considerando a estabilidade, a adequação ao projeto e que fossem de uso livre (*open source*). As principais ferramentas utilizadas para apoiar o processo de GCS proposto podem ser vistas na Tabela 4.3. Dentre essas, as três primeiras ferramentas são consideradas mais relevantes, pois, apóiam respectivamente, o sistema de controle de versões, o de mudanças e o sistema de construções. Essas ferramentas são apresentadas mais detalhadamente nesta seção, as demais foram utilizadas para viabilizar a integração entre os sistemas mencionados ou para apoiar o funcionamento do gerador e de sistemas criados. Detalhes sobre a utilização e instalação dessas ferramentas podem ser vistos nos seguintes documentos que foram elaborados durante a realização deste trabalho: Manual de Instalação e Configuração do Ambiente de GCS (Borges e Penteado, 2008a), Manual de Instalação e Configuração do Gerador GAwCRe e de Sistemas Produzidos pelo Gerador (Borges e Penteado, 2008b).

Tabela 4.3: Ferramentas de apoio utilizadas no projeto

Subversion (CollabNet, 2008a)	Sistema de controle de versões
Trac – Integrated SCM & Project Management (Trac, 2008)	Sistema de controle de mudanças
Ant 1.7 (Apache Ant, 2008)	Sistema de construções
TortoiseSVN (CollabNet, 2008b)	GUI para utilização do SVN em ambiente MS-Windows
MySQL Enterprise Server 5.0.45 (MySQL, 2008a)	SGBD
MySQL Connector/J 5.1 (MySQL, 2008b)	Driver de conexão JDBC
Eclipse (Eclipse, 2007)	IDE
JUnit (JUnit, 2007)	Framework para criação de testes de unidade
SVNAnt 1.1	*Plugin* para a integração entre o Subversion e o Ant
Tomcat (Apache Tomcat, 2008)	Servidor Web para Servlet e JSP
Java version 1.5	Java SE Development Kit (JDK)

Sistema de Controle de Versões

Os requisitos considerados para a escolha do sistema de controle de versões foram:

- Deve apoiar a identificação, o armazenamento e o gerenciamento dos ICs durante todo o ciclo de vida do software;

- Deve manter um histórico acessível de todas as alterações realizadas;
- Deve ser capaz de criar e gerenciar diferentes ramos de desenvolvimento simultâneos;
- Deve permitir a recuperação das configurações desejadas em uma linha de tempo;
- Deve ser de uso livre (*open source*);
- Deve possuir a capacidade de integração a outras ferramentas livres necessárias, como de controle de mudanças;
- Deve permitir o compartilhamento das informações armazenadas e possuir uma política de sincronização de mudanças;

O sistema de controle de versões Subversion (CollabNet, 2008a) por atender a todos esses requisitos foi escolhido. No Capítulo 2 foram apresentadas suas características bem como seu ciclo básico de funcionamento.

Sistema de Controle de Modificações

Trac, um sistema de controle de modificações e rastreamento de problemas, apresentado no Capítulo 2, foi escolhido como ferramenta de apoio ao controle de modificações. As principais vantagens dessa ferramenta são:

- Ser de uso livre (*open source*);
- Ter a capacidade de integração com o sistema de controle de versões Subversion;
- Apoiar o desenvolvimento colaborativo;
- Ser uma aplicação voltada para o ambiente Web.

O Trac utiliza um sistema de *tickets* para registrar as solicitações de modificações. Na Figura 4.8 pode se visualizar a tela onde são registrados os pedidos de mudança ou são comunicados quaisquer problemas encontrados. Cada *ticket* registrado é visível para todos os envolvidos no projeto.

Ao ser submetido, um *ticket* é classificado como novo até que uma ação lhe seja atribuída. Durante o ciclo de vida do *ticket*, seu estado é alterado e outras ações devem ser atribuídas para indicar o estado atual, como ilustrado na Figura 4.9.

- Mantido como novo por um tempo estipulado. Durante esse período, os participantes do projeto o analisam e decidem se as alterações propostas devem ou não ser realizadas. A maneira mais apropriada de se realizar as modificações propostas também são consideradas pelos participantes;

- Aceito por um usuário-desenvolvedor habilitado para resolvê-lo;
- Resolvido e liberado para a realização de testes;
- Ser julgado inválido e, após o registro da justificativa, o *ticket* é encerrado;
- Ser encerrado e marcado como resolvido;
- Um *ticket* encerrado pode ser reaberto se necessário.

É possível também associar um ou mais tickets a um milestone, assim o progresso do desenvolvimento ou manutenção podem ser acompanhados e relatórios do estado da configuração podem ser gerados com base nessas informações (Trac, 2008).

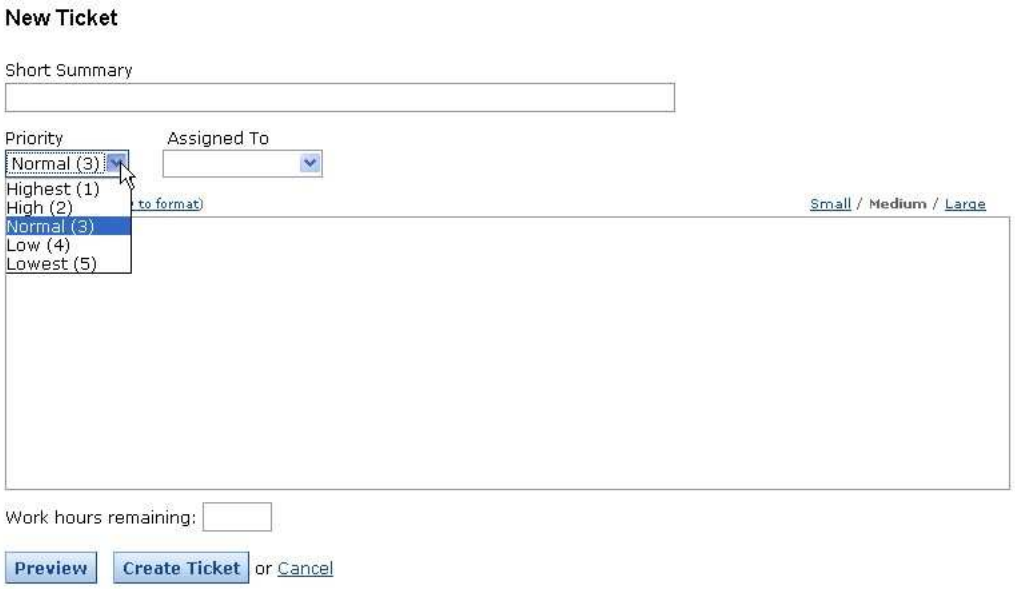

Figura 4.8: Painel de controle para criação de *tickets*

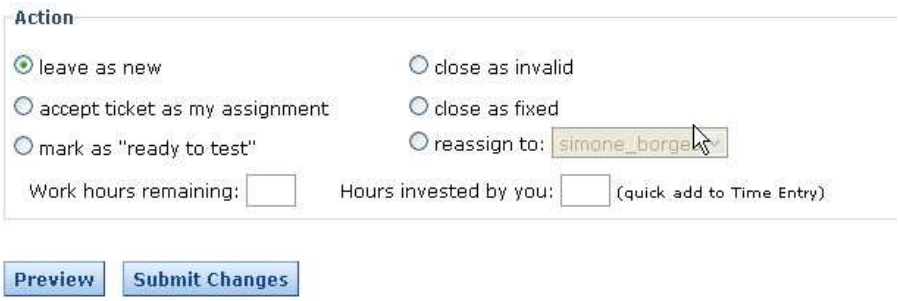

Figura 4.9: Ações que podem ser atribuídas a um *ticket*

Sistema de Gerenciamento de Construção

Um projeto de software envolve a execução de diversas tarefas como, por exemplo, controle de versões, geração de artefatos, execução de testes, preparação de distribuições, etc. A realização manual dessas tarefas consiste na execução de passos repetitivos e, normalmente, a prioridade atribuída a elas diminui gradualmente próximo à *deadlines* (Clark, 2004).

A utilização de ferramentas de automação, configuradas para executar atividades específicas, reduz a necessidade de intervenções humanas. Uma atividade típica de automação inclui a indicação dos arquivos alvo no projeto, das eventuais dependências de arquivos, de softwares ou mesmo de outras tarefas e de como a tarefa deve ser executada e qual o artefato final desejado (Prange, 2007). Assim, um único usuário pode realizar diversas tarefas em um curto prazo de tempo. Automatizações recebem uma classificação baseada no modo como são iniciadas (Clark, 2004) e podem ser:

Automação Conduzida: Acontece a cada vez que um comando é executado pelo usuário e, em seguida, executa-se um conjunto de tarefas de modo consistente e passível de repetição;

Automação Agendada: Semelhante à ação anterior, com a diferença de deixar agendado o início da atividade. Dessa forma o risco de esquecimento deixa de existir;

Automação Disparada: Tarefas de automação podem ter início após a ocorrência de algum evento importante, como por exemplo, quando um usuário executa um *commit* de seu trabalho diário. Os testes automatizados são executados e, se não forem encontrados problemas, o *commit* é realizado.

Ant possibilita a automatização de tarefas, interpretando instruções contidas em arquivos de configuração (*scripts*) escritos em XML. Esses arquivos são geralmente denominados `build.xml` e possuem uma estrutura uniforme como mostrado na Listagem 4.1. Pode ocorrer a variação de apenas alguns elementos.

Ant foi escolhido por ser independente de plataforma e, além disso, pode ser integrado a várias IDEs, como por exemplo: Eclipse (Eclipse, 2007) e Netbeans (Netbeans, 2008). Todavia, neste projeto, a integração com IDEs não foi explorada, com o objetivo de se obter um processo menos dependente de IDEs e com *scripts* de uso mais geral. Assim, as automatizações realizadas são do tipo *Conduzida*, pois o emprego dos outros dois tipos demandaria mais ferramentas além das já utilizadas.

4.4 GAwCRe com GCS

A falta de um processo de GCS dificultava o uso efetivo do GAwCRe, tanto em sua manutenção quanto na geração de artefatos. A adoção de um processo de GCS vi-

abiliza a estruturação de um ambiente de desenvolvimento de software controlado, apoiado principalmente pelos sistemas de controle de versões, de mudanças e de gerenciamento de construções. Isso possibilita a criação de aplicações no domínio da SiGLi utilizando o gerador GAwCRe em ambiente controlado e com a documentação de todos os passos relevantes. Assim, agrega-se rastreabilidade ao processo de criação de artefatos. Com a abordagem proposta, as aplicações podem ser instanciadas quantas vezes forem necessárias, e quaisquer alterações realizadas no código fonte das aplicações podem ser reutilizadas (*merge*) na nova versão.

Listagem 4.1: Estrutura básica de um *script* de automatização utilizado no Ant

```
<project name="nome_projeto" basedir="." default="alvo1">
  <target name="alvo1">
    <tarefa1 />
    <tarefa2 />
  </target>
  <target name="alvo2">
    <tarefa1 />
    <tarefa2 />
  </target>
  <target name="alvo3">
    <tarefa1 />
  </target>
</project>
```

A manutenção de sistemas legados não é trivial, bem como a de sistemas desenvolvidos sem que as práticas de engenharia de software tenham sido realizadas. Dessa forma, o processo elaborado, aprimora a manutenibilidade do gerador GAwCRe, pois colabora para que todas as solicitações de modificações sejam documentadas possam ser rastreadas.

A reengenharia de sistemas legados utilizando o gerador GAwCRe com a abordagem ARA, também se beneficia do processo de GCS. A proposta de Freitas (2006) prevê alterações unicamente no meta-modelo XML para se equiparar o sistema alvo aos requisitos do sistema legado. O processo de GCS proposto, colabora para a preservação da linguagem de padrões SiGLi, pois quaisquer adaptações para domínios conexos ao que a SiGLi foi desenvolvida podem ser criadas utilizando-se ramificações, que permite a geração de vários produtos de uma LPS.

Não há restrição do número de vezes que o sistema legado pode ser alterado utilizando a abordagem ARA (Cagnin, 2005). Com o controle de versões há a possibilidade de se criar um novo produto a partir de um já existente.

A Figura 4.10 ilustra o processo discutido neste Capítulo, ou seja, a utilização do gerador de aplicações GAwCRe no lugar de um framework na abordagem ARA e com a GCS.

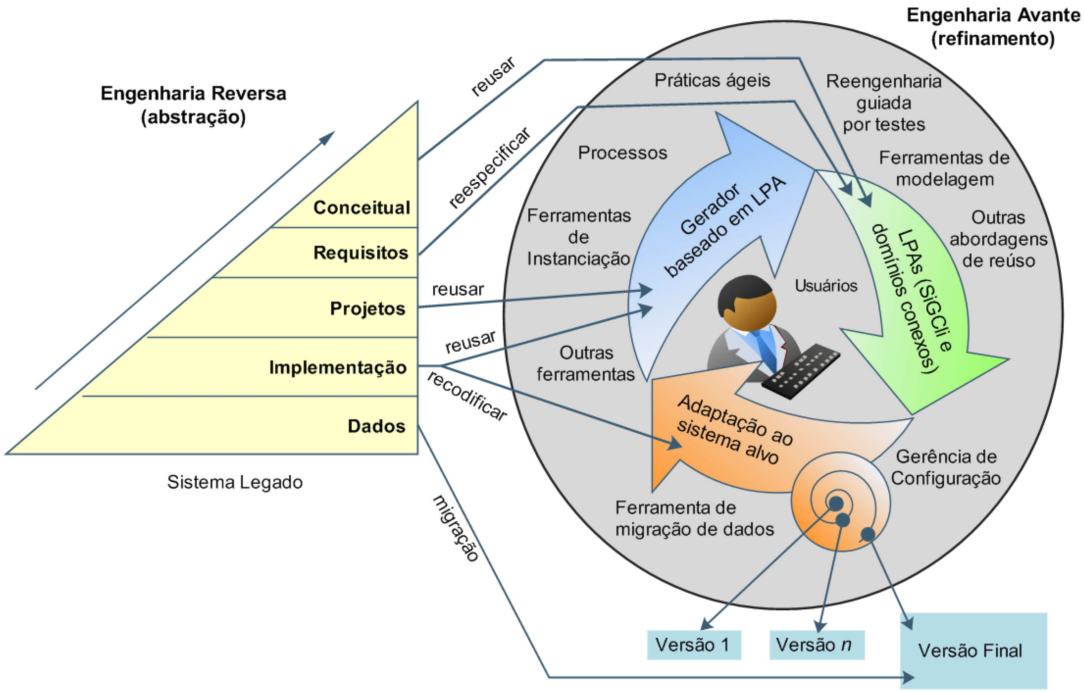

Figura 4.10: Abordagem ARA com Gerador de Aplicações e GCS

Uma ferramenta para controle de versões, denominada GREN-WizardVersionControl (Cagnin, 2005), foi elaborada para apoiar a Abordagem ARA, entretanto, essa ferramenta tem seu uso restrito ao framework GREN, sendo inviável sua utilização junto ao gerador GAwCRe. Na Tabela 4.4 é apresentado um comparativo da cobertura possível com GREN-WizardVersionControl e a abordagem apresentada neste trabalho.

Os sistemas de controle de mudanças e de controle de versões permitem a recuperação de qualquer versão desejada de qualquer artefato e a construção automatizada de configurações permite a construção de *builds* personalizados. Com isso, problemas como *Complicações na Identificação da Configuração* e *Diferentes Restrições a cada Release* podem ser resolvidos (Staples, 2004).

A estrutura de controle de mudanças utilizada permite um canal de comunicação e documentação para negociações de requisitos entre os envolvidos, podendo contribuir para solucionar o problema de *Diferentes Restrições a cada Release* (Staples, 2004).

Modificações nas similaridades (Staples, 2004) são propagadas para todos os sistemas gerados após a modificação. A criação de ramificações (*branchs*) para atender necessidades específicas pode ser a solução, criando-se variações para atender requisitos diferenciados, preservando-se a linha principal de desenvolvimento (*trunk*) ou mesmo outro ramo (*branch*).

Recursos como junção de alterações (*merge*), comparações (*diff*) e recuperação de versões anteriores (*revert*) contribuem para eliminar o problema de *Degradação da*

Linha de Produto (Staples, 2004). Quaisquer atividades de manutenção realizadas em uma linha de desenvolvimento seja ela a linha principal ou uma ramificação, podem ser incorporadas, comparadas ou revertidas utilizando-se esses recursos.

Tabela 4.4: GREN-WizardVersionControl contrastado com a abordagem de GCS proposta

	GREN-WizardVersionControl	**Plano de GCS**
Aplicável a outras ferramenta de geração de código	não	sim
Controle de *baselines*	não	sim
Controle de *releases*	não	sim
Controle de sincronização	não	sim
Recuperação de qualquer versão	não	sim
Gerencia de conflitos em tempo de instanciação	sim	não
Manutenção da base de dados	sim	não

4.5 Considerações Finais

Este capítulo apresentou uma abordagem para um processo de CV para apoiar a manutenção e evolução do gerador de aplicações GAwCRe. A falta de um processo de GCS dificultava o uso efetivo e a manutenção do GAwCRe, bem como o gerenciamento dos artefatos produzidos. A recuperação das diferentes versões do gerador e gerenciamento dessas versões por meio da adoção de um processo de GCS, baseado na norma IEEE 828-2005 e nas diretrizes propostas por Murta (2006), possibilitou a obtenção de informações sobre o histórico das manutenções realizadas nas versões do gerador.

Todas as versões do gerador foram organizadas e persistidas em um repositório controlado. A disposição dos diretórios que armazenam as versões do gerador foi organizada de acordo com o *layout* Jakarta. A utilização de tal disposição facilita a manutenção do repositório, conforme mencionado anteriormente e a automatização de tarefas utilizando ferramentas como, por exemplo, Apache Ant (2008). Outro benefício resultante é que, além da preservação (*trunk*) do GAwCRe, preserva-se também a linguagem de padrões SiGLi. Desse modo, quaisquer adaptações para domínios conexos podem agora ser criadas (*branch*), permitindo assim a exploração da variabilidade do gerador.

A fim de se obter controle adequado sobre as alterações que devem ser realizadas,

em cada uma das versões do gerador, a ferramenta Trac foi empregada. Sua utilização aprimora a rastreabilidade das alterações solicitadas. A ferramenta registra informações tanto sobre alterações realizadas quanto não realizadas; documentando as justificativas para realização ou não de determinadas solicitações.

A instalação e configuração do GAwCRe, assim como a de seus artefatos, era realizada de maneira não automatizada, que demandava muito tempo e eram propensas à introdução de erros. A ferramenta Apache Ant (2008) foi usada para automatização de algumas tarefas relacionadas à instalação e configuração do gerador. Por exemplo, a construção de *builds* para a versão `RB_03` do gerador foi automatizada. O *script* implementado "extrai" os arquivos pertencentes ao gerador, como por exemplo, classes e bibliotecas, e os organiza de forma que possam ser utilizados por IDEs como o Eclipse. Essa atividade é necessária, pois a disposição dos diretórios armazenados no repositório é diferente da requisitada pela maioria das IDEs.

O plano de GCS definido também é considerado um IC, desse modo cada versão do gerador contém o subdiretório documentacao que, conforme descrito no Apêndice A, pode ser usado para armazenar versões modificadas do plano "original" que se encontra armazenado no diretório: `...\branches\RB_03\documentacao`. Uma vantagem resultante dessa organização é que o plano de GCS pode ser "ajustado" de acordo com as necessidades de cada versão do gerador, de forma que a evolução de ambos é facilmente relacionada e gerenciada. É importante ressaltar que o plano de GCS deve ser disponibilizado de maneira *stand-alone* ou anexado a outro documento do projeto, devidamente identificado de modo a não deixar dúvidas sobre sua localização ou sobre seu conteúdo. Essa é uma determinação da norma IEEE 828-2005.

O Capítulo seguinte apresenta um estudo de caso que descreve a reengenharia de dois sistemas legados, pertencentes a domínios conexos ao abordado pela SiGLi. Nesse estudo de caso descrevem-se as adaptações realizadas no gerador GAwCRe a fim de produzir as aplicações que implementam a funcionalidade requisitada pelos domínios conexos. As modificações necessárias foram realizadas em ambiente controlado e com a documentação de todos os passos relevantes.

CAPÍTULO
5

Aplicação da Proposta em Estudos de Caso

5.1 Considerações Iniciais

A falta de apoio de técnicas de GCS na utilização do Processo Ágil de Reengenharia Utilizando Geradores de Aplicações não permitia que as versões modificadas do gerador e também dos artefatos gerados pelo GAwCRe fossem posteriormente recuperadas. Como solução, uma abordagem de GCS foi elaborada a fim de permitir o controle e rastreamento das adaptações necessárias para viabilizar a evolução e manutenção tanto do gerador de aplicações quanto dos artefatos gerados. Um ambiente de GCS composto por sistema de controle de versões, controle de mudanças e gerenciamento de construções foi configurado utilizando ferramentas de apoio *open source*, como descrito no Capítulo anterior. A abordagem de GCS elaborada, também foi utilizada para controlar e documentar as atividades de manutenção corretiva, que foram necessárias, para sanar os problemas comentados no Capítulo 4 e para permitir o uso efetivo do gerador GAwCRe (Borges e Penteado, 2008c).

Neste Capítulo é apresentado um estudo de caso a fim de exemplificar a abordagem de reengenharia utilizando GCS. Dois sistemas foram selecionados para realização da reengenharia:

- *Fisiosoftware* (Biomanager, 2008a) um sistema de atendimento para clínicas de fisioterapia, pertencente ao domínio do GAwCRe;
- *Psychologist Software* (Biomanager, 2008b) sistema de atendimento a clínicas de

psicologia, pertencente a domínio conexo ao do GAwCre.

Ambos os sistemas foram obtidos via Internet. Esses sistemas permitem o gerenciamento de clínicas nos respectivos domínios citados, e apóiam o controle do atendimento de pacientes, registro e acompanhamento de consultas, tratamentos, exames e também o controle de finanças.

Este capítulo está organizado da seguinte maneira: na Seção 5.2 é apresentado o estudo de caso realizado, tendo como exemplo o sistema *Fisiosoftware*, e na Seção 5.3 são comentadas as diferenças existentes no sistema *Psychologist Software* em relação ao estudo de caso com o sistema *Fisiosoftware*. Na Seção 5.4 são feitas as considerações finais.

5.2 Estudo de Caso: Sistema *Fisiosoftware*

O processo para realizar a reengenharia utilizando gerador de aplicações é composto por quatro fases: Concepção, Elaboração, Construção e Transição. Cada uma dessas fases são compostas, por sua vez de atividades específicas. A seguir, essas fases são descritas e exemplificadas por meio da reengenharia do sistema *Fisiosoftware*.

5.2.1 Fase de Concepção

A finalidade da fase de Concepção é permitir que o engenheiro de software entenda o domínio da ferramenta de geração dos artefatos e o domínio do sistema legado e avalie se a reengenharia é possível (Cagnin, 2005; Freitas, 2006).

Familiarizar-se com o Domínio do Gerador

Foi feito o *check out* da versão denominada `RB_03` a partir do repositório do gerador GAwCRe. Utilizando o IDE Eclipse uma estrutura de projeto foi criada para satisfazer a disposição dos arquivos fonte pertencentes ao GAwCRe.

Algumas atividades de manutenção corretiva foram realizadas a fim de que o gerador fosse instanciado (Borges e Penteado, 2008c). Em seguida, foram criadas aplicações hipotéticas nos domínios atendidos pelo gerador com a finalidade de familiarização com a sua interface, sua usabilidade e os tipos de artefatos gerados. Os passos necessários para a criação desses sistemas foram documentados e as informações coletadas foram utilizadas para compor uma manual para guiar a criação de futuras aplicações utilizando o gerador GAwCRe (Borges e Penteado, 2008b).

Buscar Sistemas Legados

O sistema *Fisiosoftware* (Biomanager, 2008a) foi o primeiro a ser selecionado. Esse sistema permite o gerenciamento de clínicas e consultórios de fisioterapia, sua fun-

cionalidade inclui funções de cadastro de pacientes e anamnese, agendamento de consultas, controle de contas a pagar e a receber e *backup* dos dados.

Verificar o Domínio do Sistema Legado e Observar o Domínio do Sistema Legado em Relação ao Gerador

A Figura 5.1 foi construída para elencar as macro funções identificadas, subdivididas em funções e os respectivos padrões da SiGCli que podem atender à implementação dessas funções para o sistema *Fisiosoftware*.

Fisiosoftware			SiGCli	
Macro Funções	Funções		Nº	Padrão
Arquivo	MS-Word			n.a.
	Paint			n.a.
Backup	Filzip			n.a.
	Brazip			n.a.
Agenda	Agendar Consultas		5	Agendar Atendimento
	Configurar Horário da Agenda		5	Agendar Atendimento
	Avisos	Aniversariantes		n.a.
		Gerais		n.a.
	Calendário			n.a.
Paciente	Anamnese	Anamnese do Paciente	7	Realizar Acompanhamento
		Cadastrar Perguntas		n.a.
	Cadastro do Paciente		1	Identificar Paciente
	Ficha Clínica		2	Definir Serviços
			3	Realizar Vendas
			4	Processar Guias
			6	Identificar Atendentes
			7	Realizar Acompanhamento
Textos	*Modelos diversos de documentos em formato MS-Word.*			n.a.
Financeiro	Contas a Pagar		9	Controlar Faturamento
	Contas a Receber		9	Controlar Faturamento
	Extrato de Movimentação Financeira			n.a.
	Histórico Financeiro			n.a.
Senha	Modificar a Senha			n.a.
Sobre				-
Sair				-
Legenda		n.a. – não atende		a.p. – atende parcialmente

Figura 5.1: Comparação das funções do sistema *Fisiosoftware* e padrões da SiGLi

As funções: Arquivo, Backup, Textos e Senha existentes no sistema *Fisiosoftware* não são atendidas pelos padrões da SiGCli. As funções Avisos e Calendário, relativas à macro função Agenda não são apoiadas pelo padrão de número 5, Agendar Atendimento.

A função Anamnese é atendida parcialmente pelo padrão de número sete, denominado *Realizar Acompanhamento*. No gerador, quando esse padrão é aplicado, perguntas relativas ao domínio de clínicas de fisioterapia, terapia ocupacional e/ou educação física são geradas juntamente com a aplicação. Essas perguntas encontram-se registradas no mapeamento XML, todavia não existe a função de edição das perguntas existentes ou o cadastro de novas para personalizar uma determinada clínica.

Assim, dos nove padrões que compõem a SiGCli, somente o padrão de número oito – *Realizar Compras*, não é utilizado nesta aplicação.

5.2.2 Fase de Elaboração

Durante a fase de Elaboração, todas as funções do sistema legado, que foram identificadas na fase de Concepção, foram estudadas para permitir que o engenheiro de software decidisse a ordem de implementação dessas funções. O número de iterações e a quantidade de requisitos que foram atendidos a cada iteração foram decididos pelo engenheiro de software devido à ausência de usuários reais do sistema, que de outra forma seriam incentivados a colaborar nessa atividade. Para apoiar essa decisão, o diagrama de casos de uso do sistema foi elaborado. Além de colaborar para a compreensão das funções do sistema legado, o diagrama foi posteriormente, armazenado no repositório do sistema *Fisiosoftware* como contribuição à documentação do sistema. Também foi elaborada a documentação dos casos de testes.

Desenvolver o Diagrama de Casos de Uso

O diagrama de casos de uso foi elaborado utilizando o *plugin* EclipseUML 2007 Free Edition (Omondo, 2008) para o IDE Eclipse e é exibido na Figura 5.2, na Figura 5.3 é exibida a especificação do caso de uso. Como o processo de reengenharia é incremental e iterativo, os requisitos foram priorizados para serem implementados a cada iteração. Para fins de exemplificação do processo de reengenharia, foi selecionado o caso de uso *Cadastrar Pacientes*.

Elaborar os Casos de Teste

Para realização dos casos de testes foram elaborados gabaritos, sendo que critérios de teste funcional foram utilizados para verificar a funcionalidade do sistema legado. De acordo com os atributos e funções do caso de uso priorizado (*Cadastrar Pacientes*) o gabarito foi preenchido e é exibido, parcialmente, Figura 5.4.

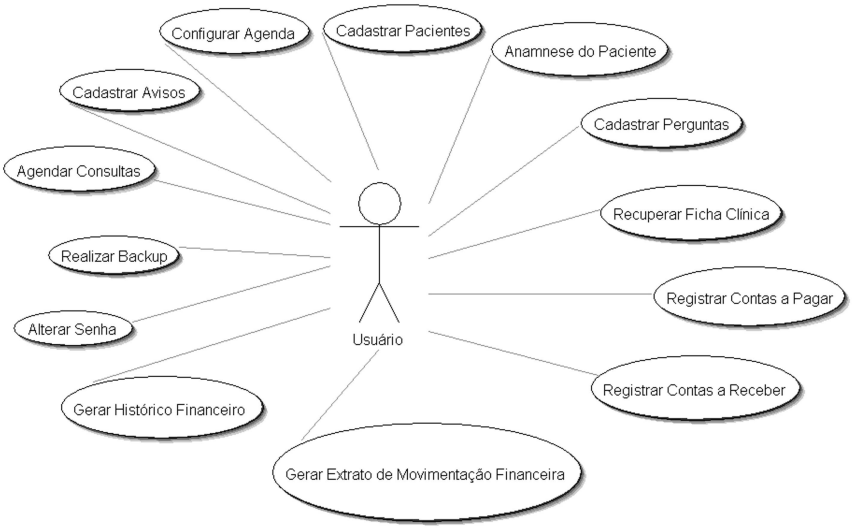

Figura 5.2: Diagrama de Casos de Uso

```
Caso de uso: CADASTRAR PACIENTES
Descrição: possibilitar que o usuário realize o
cadastro das pessoas que freqüentam a clínica.

1. Paciente fornece seu nome;

2. Paciente não cadastrado;

Fluxo Normal - Inclusão
   2.1.     O código do paciente é gerado automaticamente
            pelo sistema;
   2.2.     O paciente fornece seus dados;
   2.3.     Criar paciente;

Fluxo Alternativo - Alteração
   2.4.     Paciente já cadastrado
      2.4.1.      Paciente deseja alterar dados;
      2.4.2.      Paciente fornece dados a serem alterados;
      2.4.3.      Alterar Paciente;
      2.4.4.      Encerrar caso de uso

Fluxo Alternativo - Exclusão
      2.4.5.      Paciente será excluído;
      2.4.6.      Excluir Paciente;
      2.4.7.      Encerrar caso de uso.

      Fluxo Alternativo - Impressão
      2.4.8.      Paciente será impresso;
      2.4.9.      Imprimir Paciente;
 Encerrar caso de uso.
```

Figura 5.3: Caso de Uso *Cadastrar Pacientes*

5.2.3 Fase de Construção

Na fase de Construção, o gerador GAwCRe foi instanciado e os padrões apresentados na Figura 5.1 foram aplicados para a obtenção do sistema alvo.

		Classes de Equivalência		Casos de Testes	
Atributos	**Descrição Banco**	**Classes Válidas**	**Classes Inválidas**	**Casos de Teste**	**Saída Esperada**
Código ou Nome na busca)	Texto 50	Texto ou vazio	----	Código ou Nome cadastrado	Paciente aparece na lista
				Vazio	Todos os pacientes aparecem
Código do Paciente	Numeração automática	Gera Código	Não gera código	Verificar código	Código gerado automaticamente
Data do Cadastro	Texto 12	Data Atual	Data ! = Atual	Verificar Data	Data Atual
Paciente	Texto 50	Texto >= 1 Texto <= 50	Texto < 1 Texto > 50	Texto <= 50 Texto >= 1	Sucesso
...	...	...	...	...	...

Figura 5.4: Documentação parcial dos casos de testes da função *Cadastrar Paciente*

Criar o Sistema Alvo no Paradigma Orientado a Objetos

Após a seleção dos padrões para compor a aplicação, os artefatos foram gerados a partir do menu do instanciador.

Scripts de criação do banco de dados: O menu Gerar SQL aciona o módulo gerador de scripts SQL que gera a estrutura do banco de dados MySQL para a aplicação. Esses arquivos são armazenados em um diretório denominado `sqlFiles`;

Classes Java: O menu Gerar Código aciona o módulo gerador das classes Java (*Beans*) que têm as regras de negócios da aplicação. As classes são criadas em um diretório que possui o mesmo nome da aplicação sendo considerada, nesse caso o diretório chama-se *Fisiosoftware*;

Interface Web da aplicação: O menu Gerar Interface aciona o módulo gerador de interfaces JavaServer Pages (JSP) (Sun Microsystems, Inc., 2008) que cria as interfaces Web em que o usuário final interage com a aplicação e que são visualizadas no navegador (*browser*). As páginas JSP são armazenadas no diretório `JSPFiles`.

Ciclo básico de operação do sistema de controle de versões:

1. Um diretório denominado `Fisiosoftware` foi criado no repositório produto, esse repositório foi criado especificamente para o armazenamento dos sistemas gerados pelo GAwCRe de acordo com o plano de GCS elaborado no Capítulo 4;

2. Foi feito o *import*[1] dos artefatos gerados (sistema alvo) para o repositório do sistema de controle de versões, até esse momento os artefatos não se encontravam sob controle de versões;

3. A cópia local do sistema alvo gerado foi apagada, pois não era mais necessária;

4. Foi feito o *check out* do sistema alvo para um espaço de trabalho no computador local do engenheiro de software. Na Figura 5.5 o número 1 indica o endereço local onde foi criado o espaço de trabalho, e o número 2 indica os três diretórios que contém os artefatos gerados pelo GAwCRe e que foram obtidos mediante a ação de *check out*.

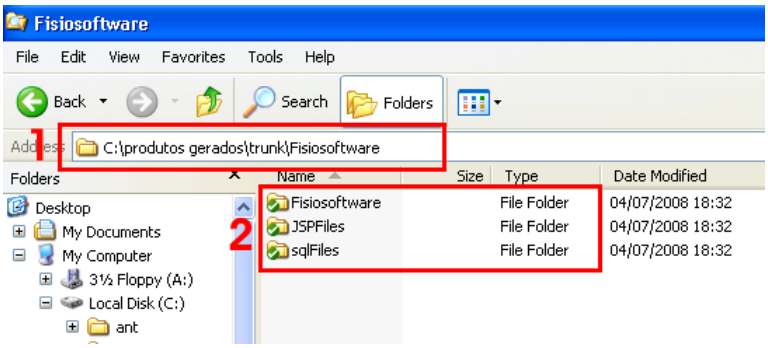

Figura 5.5: Espaço de trabalho contendo artefatos gerados pelo GAwCRe

- Dois diretórios auxiliares foram criados no repositório do sistema *Fisiosoftware*, e são destacados com o símbolo * na Figura 5.6. O diretório documentação foi utilizado para armazenar os arquivos de documentação, modelagem e testes produzidos durante as atividades de reengenharia e o diretório *build*, para armazenar *scripts* de automação para a criação de distribuições da aplicação gerada.

Executar os Casos de Testes no Sistema Alvo

Um *script* de automação foi elaborado para criação da estrutura de diretórios que é necessária ao funcionamento das aplicações geradas pelo GAwCRe. As seguintes atividades relacionadas na Tabela 5.1 são executadas:

Após a execução do passo anterior e com a estrutura de diretórios necessária para receber as aplicações geradas pelo GAwCRe criada e configurada, pôde se verificar o status do servidor, acessando o endereço: `http://localhost:8080/dev/`, a tela de boas vindas ilustrada na Figura 5.7 foi apresentada.

[1] *Import* é o termo utilizado pelo Subversion para designar a ação de persistir ICs de um projeto pela primeira vez no repositório.

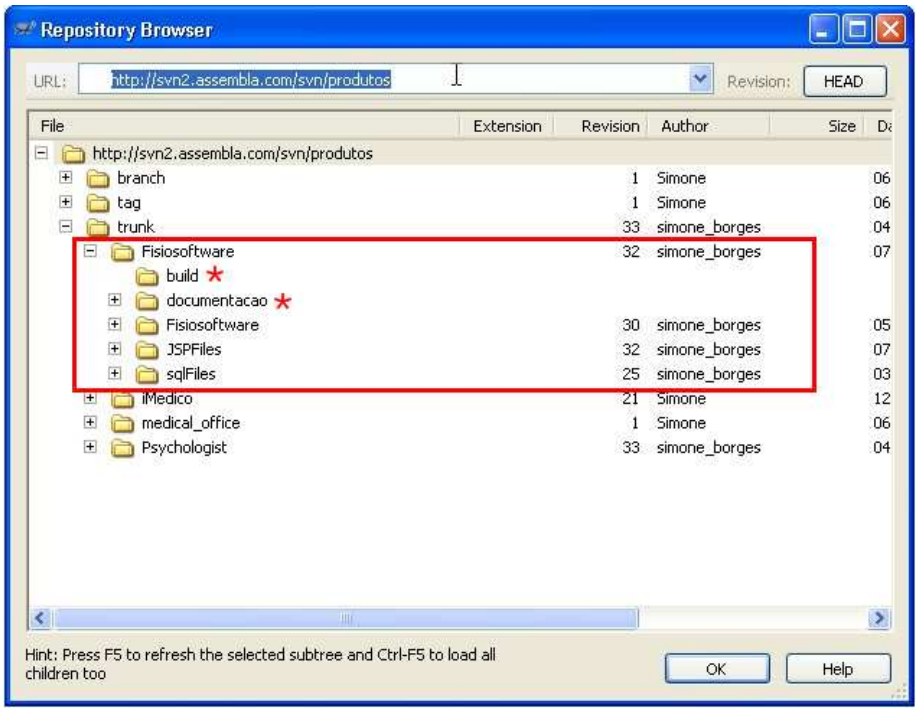

Figura 5.6: Localização do diretório do sistema *Fisiosoftware*

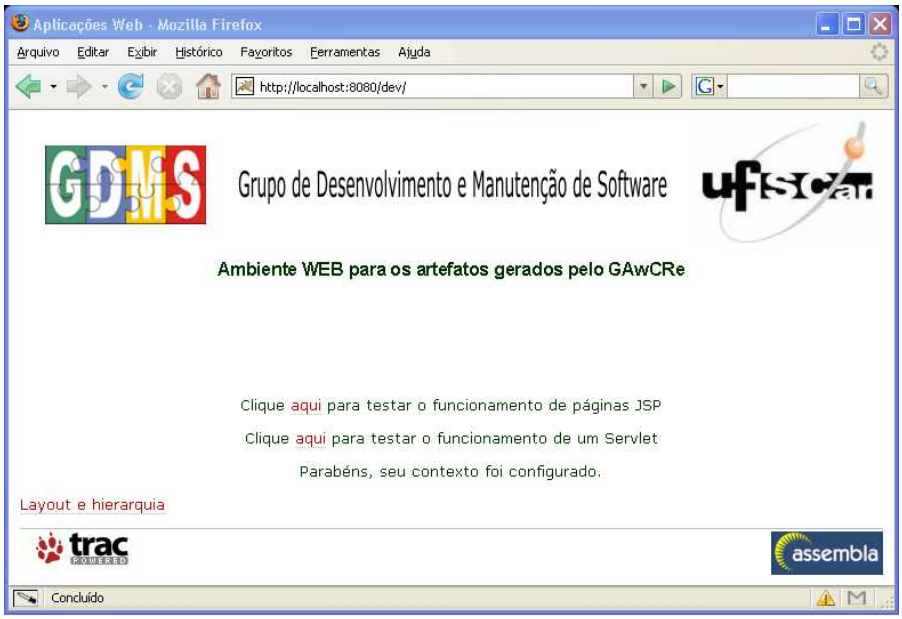

Figura 5.7: Tela de apresentação do ambiente Web

A instalação de artefatos gerados pelo GAwCRe ainda não ocorre de forma automatizada, portanto foi realizada manualmente pelo engenheiro de software. Assim, após a configuração do ambiente Web, o sistema alvo foi instalado de acordo com os seguintes passos:

Tabela 5.1: Tarefas executadas pelo *buildfile* que configura o ambiente Web

Cria todos os diretórios necessários e na hierarquia necessária para o funcionamento das aplicações geradas pelo GAwCRe.
Disponibiliza arquivos de orientação do tipo `leiame.txt`.
Disponibiliza os arquivos `dev.xml` e `web.xml`, já configurados para o funcionamento do Tomcat.
Realiza o *check out* do conteúdo dos diretórios `comum` e `pl` para a devida localização.
Realiza o *Export* das bibliotecas (.jar) necessárias para que as aplicações geradas.
Compila os arquivos .java no diretório `pl`, disponibilizando o *package* para as aplicações.

1. As classes Java armazenadas no diretório `Fisiosoftware` foram compiladas e copiadas para o diretório `C:\dev\web\WEB-INF\classes\fisiosoftware`, no servidor Web;

2. O conteúdo do diretório `JSPFiles` foi copiado para `C:\dev\web\fisiosoftware`;

3. A ferramenta MySQL Query Browser, exibida na Figura 5.8, foi utilizada para executar os *scripts* de criação da base de dados e respectivas tabelas do sistema alvo;

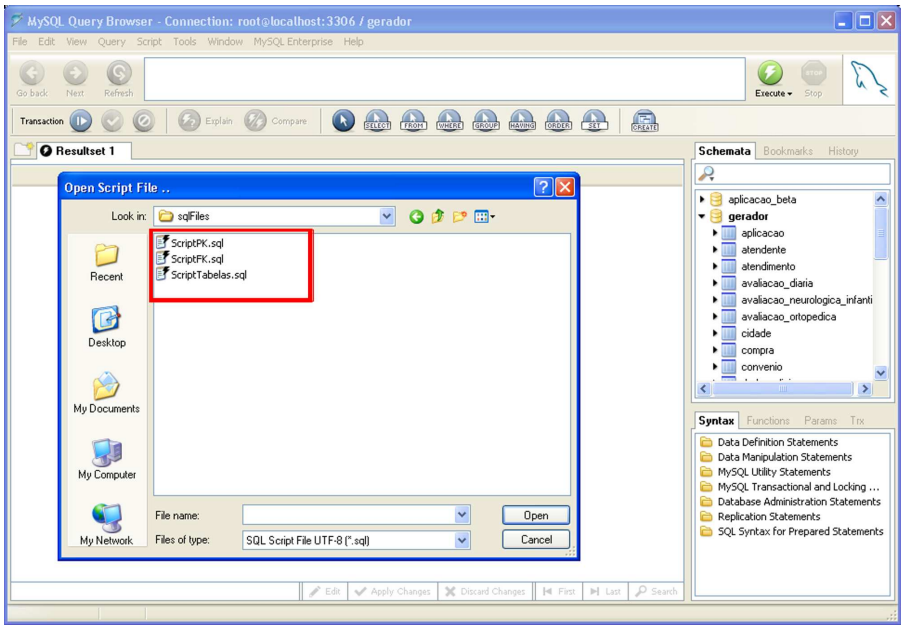

Figura 5.8: Tela da ferramenta MySQL Query Browser

Concluídos esses passos, a primeira versão produzida do sistema alvo foi instalada. Utilizando um navegador Web (*browser*) o sistema alvo foi executado, a Figura 5.9 exibe a tela da função *Cadastro de Paciente* tal como foi construída pelo gerador GAwCRe nessa primeira iteração.

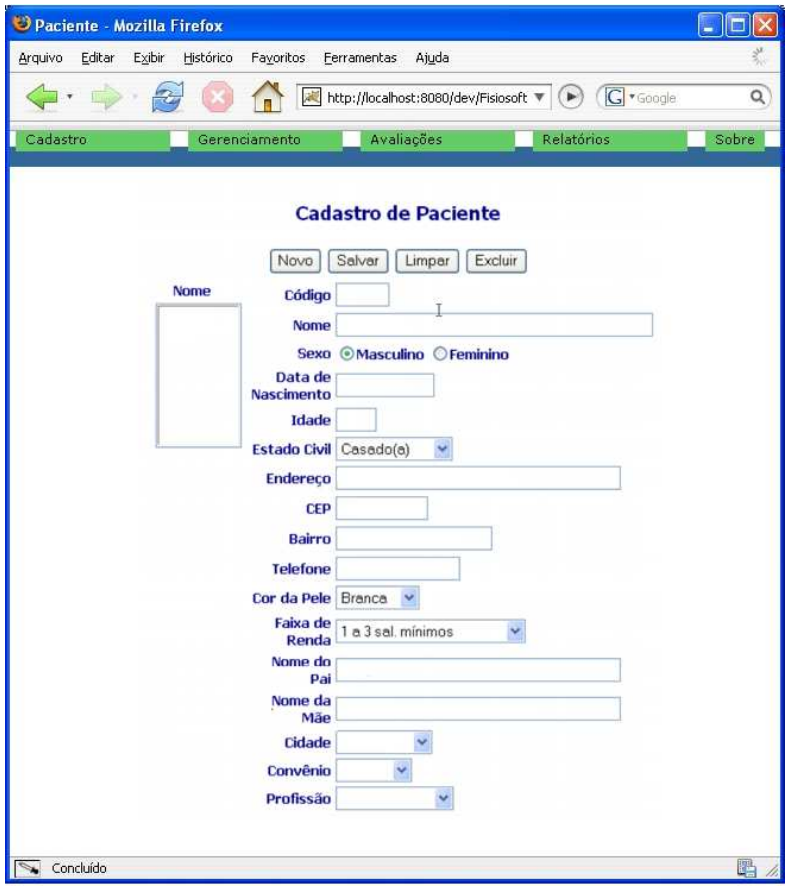

Figura 5.9: Tela do sistema alvo exibindo a função *Cadastro de Paciente*

Os casos de testes elaborados e aplicados ao sistema legado durante a atividade *Elaborar os casos de testes* foram aplicados ao sistema alvo. O objetivo dessa atividade é observar se a funcionalidade do sistema alvo gerado equivale à do sistema legado. Observou-se uma divergência entre os atributos dos dois sistemas com a ausência de atributos que constam nos requisitos e que não foram gerados e também atributos desnecessários. Na Figura 5.10 são ilustradas as tela da função *Cadastro de Pacientes* contrastadas nos dois sistemas. A tela de número 1 pertence ao sistema legado e a de número 2 ao sistema alvo. Os campos cujos atributos divergem entre os sistemas foram sublinhados em ambas as telas. Na Tabela 5.2, os campos ausentes são elencados à esquerda e à direita os que não são necessários.

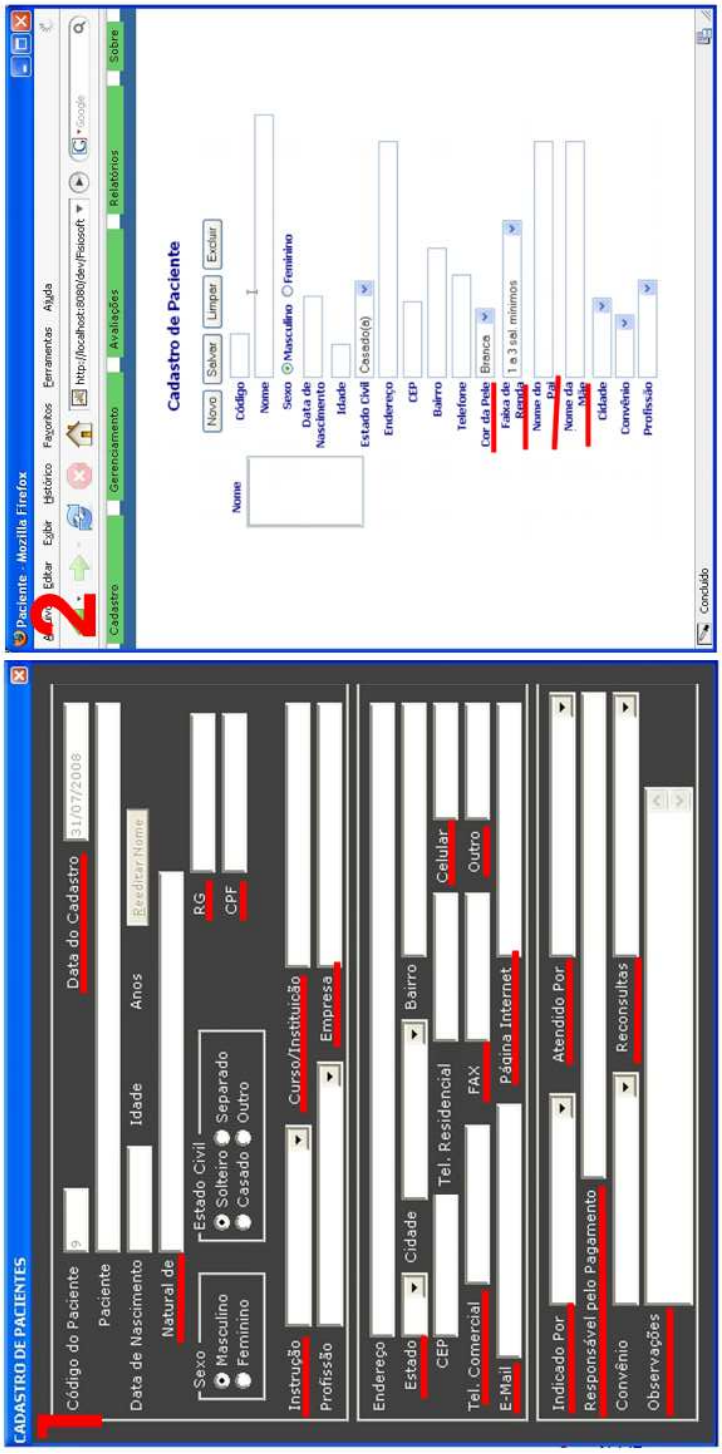

Figura 5.10: *Cadastro de Pacientes*

Adaptar o Sistema Alvo

Na abordagem ARA, requisitos específicos não cobertos pelo framework GREN são implementados diretamente no sistema alvo (Cagnin, 2005). A reengenharia ágil uti-

Tabela 5.2: Lista de campos divergentes

NECESSÁRIO	DESNECESSÁRIO
Data de Cadastro	Cor da Pele
Natural de	Faixa de Renda
RG	Nome Pai
CPF	Nome Mãe
Instrução	
Curso	
Empresa	
Tel. Comercial	
Celular	
FAX	
Outro	
E-mail	
Página Internet	
Indicado Por	
Atendido Por	
Responsável pelo Pagamento	
Reconsultas	
Observações	

lizando geradores de aplicações (Freitas, 2006) propõe a realização dessas alterações no mapeamento XML do gerador (Pazin, 2004). Três soluções foram avaliadas a fim de se obter a equiparação entre os sistemas:

1. Acrescentar e/ou eliminar atributos pela edição do código fonte do sistema alvo. Como o sistema produzido encontra-se gerenciado por um sistema de controle de versões, quaisquer alterações são rastreadas e controladas. Caso seja necessário, o sistema pode ser novamente gerado e utilizando recursos de comparação de diferenças (*diff*) e junção (*merge*) as alterações efetuadas diretamente no código fonte do sistema alvo da versão anterior podem ser integradas à nova versão, sem prejuízo do sistema, nem perda de trabalho já realizado. Essa abordagem apresenta risco inerente que é a ausência de conhecimento e de experiência do pessoal envolvido na reengenharia (Rosenberg, 1996), a qual pode levar à realização de alterações indevidas. Além disso, considerando que geradores de aplicações são ferramentas de software que produzem artefatos a partir de especificações de alto nível, um processo de desenvolvimento baseado em geradores preconiza que eventuais manutenções dos artefatos gerados sejam sempre e integralmente realizadas em sua especificação e não nos seus arquivos de implementação (código fonte gerado);

2. Acrescentar e/ou eliminar atributos com a alteração das respectivas classes mapeadas no arquivo XML. Assim, na próxima iteração, o sistema alvo pode ser gerado com os mesmos atributos que os do sistema legado. Entretanto, o domínio da SiGCli seria alterado e essa nova versão do gerador perderia a capacidade de gerar aplicações para clínicas de reabilitação (Staples, 2004);

3. Outra abordagem é manter os atributos não previstos nos requisitos e considerá-los como uma evolução do sistema que passou por reengenharia. Assim, o sistema alvo teria os requisitos inicialmente definidos (oriundos do sistema legado) e novas funções oriundas da SiGCli. Entretanto, existem alguns riscos associados, tais como recuperação de informação sem utilidade e elicitação de objetos incorreta (Rosenberg, 1996).

Para obter a equivalência de requisitos da função *Cadastro de Paciente*, a opção 2 foi escolhida e essa decisão foi registrada na ferramenta de controle de mudanças. Dessa forma, para acrescentar os atributos necessários e que não foram criados, *tags* e atributos de *tags* foram inseridos na especificação XML do padrão (1) *Identificar Paciente*, como exibido na Listagem 5.1. E os trechos de código responsáveis pela implementação dos atributos desnecessários foram comentados, no sentido em que foi acrescentada *tag* da linguagem XML que indica que o trecho de código considerado refere-se a um comentário.

Listagem 5.1: Inclusão de atributos na especificação em XML do padrão (1) *Identificar Paciente*

```
1  ...
2  <atributo tipo="String" tamanho="15" tipoCampo="text"
3  nome="Telefone Residencial"> telefone_residencial</atributo>
4
5  <atributo tipo="String" tamanho="15" tipoCampo="text"
6  nome="Celular">celular</atributo>
7
8  <atributo tipo="String" tamanho="15" tipoCampo="text"
9  nome="Telefone Comercial">telefone_comercial</atributo>
10
11 <atributo tipo="String" tamanho="15" tipoCampo="select"
12 nome="Grau de Instrucao">
13 grau_de_instrucao
14 <valor exibe="Ensino Fundamental" grava="ensino_fundamental"></valor>
15 <valor exibe="Ensino Medio" grava="ensino_medio"></valor>
16 <valor exibe="Ensino Superior" grava="ensino_superior"></valor>
17 <valor exibe="Pos-Graduacao" grava="pos_graduacao"></valor>
18 </atributo>
19 ...
```

A especificação XML alterada recebeu um nome significativo em relação ao projeto de reengenharia do sistema *Fisiosoftware* e foi armazenada no repositório do projeto. Novos projetos de reengenharia de sistemas pertencentes ao domínio do gerador e com requisitos semelhantes a esse podem fazer uso dessa especificação. Além disso, o documento XML que contém o mapeamento original da SiGCli é preservado. A Figura 5.11 exibe a tela de inicialização na qual é possível indicar a localização e em seguida selecionar o documento necessário ao mapeamento da XML (Borges et al., 2008). O gerador GAwCRe foi novamente instanciado, dessa vez utilizando o mapeamento XML adaptado para atender o sistema *Fisiosoftware*, como demonstrado na Figura 5.11 e uma nova versão do sistema alvo foi gerada e instalada no servidor Web. A Figura 5.12 ilustra a tela do *Cadastro de Paciente* dessa nova versão.

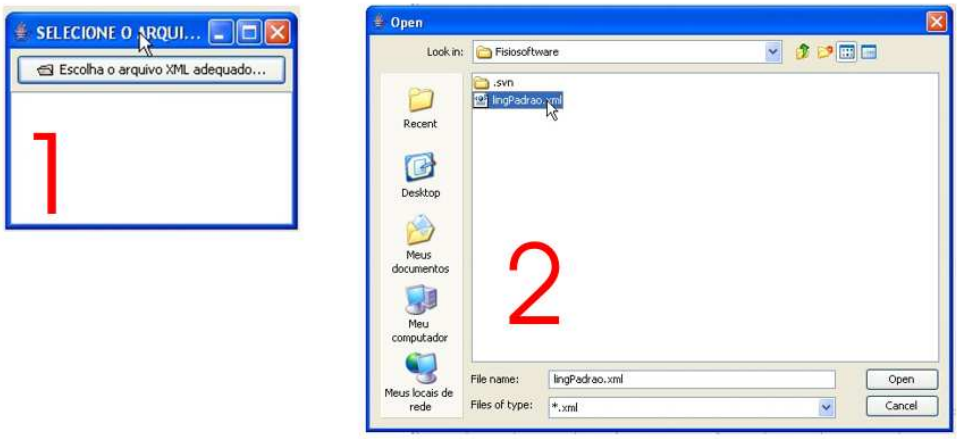

Figura 5.11: Tela de inicialização do gerador

Os casos de testes foram novamente aplicados para assegurar se a funcionalidade entre os sistemas é equivalente. Ao fim dessa iteração a função *Cadastro do Paciente* foi obtida no sistema alvo com a mesma funcionalidade prevista no sistema legado. Essa nova versão foi persistida no repositório do projeto.

5.2.4 Fase de Transição

As atividades da fase de Transição não foram realizadas, pois a reengenharia do sistema não foi completa.

As funções Arquivo, Backup, Textos e Senha não são inerentes à SiGCli, pois não fazem parte de seu domínio. Infere-se que essas funções podem pertencer a domínios conexos como o de segurança, sendo necessária a sua implementação diretamente no sistema alvo.

Durante a realização do processo de reengenharia, atividades de manutenção perfectiva foram realizadas no gerador GAwCRe com a finalidade de permitir a customiza-

ção da interface das aplicações geradas e para aprimorar a inteligibilidade do código fonte do gerador. Essas atividades encontram-se descritas em (Borges e Penteado, 2008c).

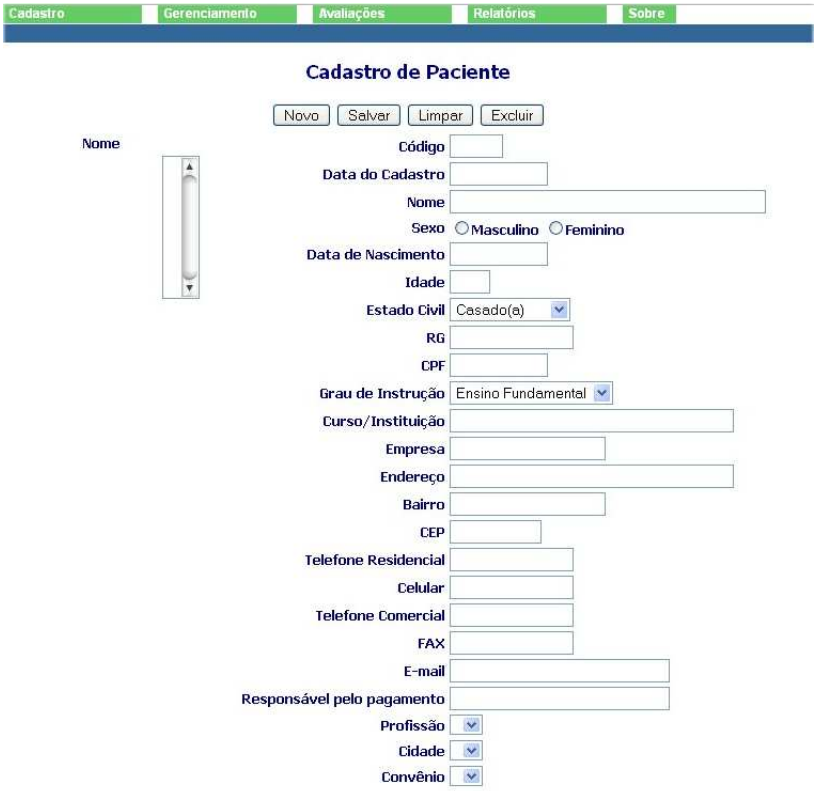

Figura 5.12: Nova versão do sistema alvo

5.3 Estudo de Caso: Sistema *Psychologist Software*

O *Psychologist Software* é um sistema para gestão de consultórios de psicologia, sua funcionalidade viabiliza a administração tanto do atendimento clínico quanto da movimentação financeira. Este sistema trabalha com cadastro de pacientes, ficha clínica, agendamento de consultas, anamnese, contas a pagar, contas a receber, conta do paciente e back up dos dados. Uma ficha clínica é gerada para cada paciente possibilitando o registro e armazenamento de todos os dados, gerando um histórico de sua ficha clínica.

A execução das atividades realizadas durante as fases de Concepção e Elaboração é idêntica à realizada no sistema *Fisiosoftware*. Durante a realização da atividade *Executar os casos de testes* no sistema alvo, observou-se que os padrões da linguagem SiGCli não cobrem totalmente o domínio dessa aplicação. Além disso, novamente foi observada a presença de classes e atributos que foram gerados e que não são neces-

sários para a aplicação. Não há como escolher se essas classes serão ou não geradas, pois elas fazem parte de padrões cuja aplicação é obrigatória quando utilizando a linguagem SiGCli.

Novamente foi necessário ponderar sobre qual a abordagem mais adequada para continuar o processo de reengenharia para esse sistema específico. Algumas diferenças entre os atributos nas classes do sistema alvo e o legado podem ser resolvidas removendo os atributos desnecessários e incluindo os ausentes, alterando-se diretamente as respectivas classes mapeadas no arquivo XML. Dessa forma, a aplicação seria gerada com os atributos iguais aos do sistema *Psychologist Software*, entretanto o domínio da SiGCli seria alterado, e essa nova versão do gerador perderia a capacidade de gerar aplicações para clínicas de reabilitação.

Outra solução possível é não remover das classes da SiGCli os atributos desnecessários para clínicas de psicologia e acrescentar os atributos que constam nos requisitos, preservando o domínio original. Desse modo, a aplicação seria gerada com atributos de ambos os domínios em suas classes. Também seria necessário eliminar atributos desnecessários, editando o código fonte do sistema alvo.

Foi seguida a abordagem proposta por Freitas (2006) e as classes ausentes foram acrescentadas sem alteração do domínio da SiGCli, optou-se por modificar somente as variantes dos padrões e criar novas variantes para incluir tais classes. As classes obrigatórias da SiGCli, e que são desnecessárias para o sistema *Psychologist Software* continuam a ser geradas para que o domínio da SiGCli não seja afetado.

Somente apoiada pelas soluções citadas, não foi possível realizar a expansão do domínio da SiGCli para clínicas de psicologia. Tal como seus padrões estão estruturados, não é possível a realização de alterações no mapeamento XML sem o risco de se degradar a linguagem.

5.4 Considerações Finais

A manutenção de sistemas legados não é trivial, bem como a de sistemas desenvolvidos sem que as atividades de engenharia de software tenham sido realizadas. A utilização do gerador GAwCRe para o desenvolvimento de aplicações pertencentes a domínios conexos ao seu, demandou diversas modificações executadas *ad hoc* (Freitas, 2006).

Quando da utilização desse gerador, para avaliar tais modificações, observou-se que eram necessárias atividades de manutenção corretiva para permitir seu uso efetivo e durante o processo de reengenharia, constatou-se a necessidade de se realizar também algumas atividades de manutenção perfectiva. Problemas como, a falta de mecanismo para controle de versões no gerador, o compartilhamento da base de dados das aplicações geradas e problemas de escopo em alguns métodos prejudicavam

o funcionamento do gerador.

O processo de GCS definido no Capítulo 4 apoiou todas as atividades realizadas. Dessa forma, após a realização da atividade *Criar o sistema alvo no paradigma orientado a objetos*, a primeira versão do sistema alvo deve ser colocada sob o controle de versões. Em seguida, cada iteração deve possuir um registro das atividades no sistema de controle de mudanças. Assim, agrega-se rastreabilidade a todas as alterações realizadas que são passíveis de serem revertidas, reproduzidas, revisadas e alteradas.

Para a realização da atividade *Adaptar o sistema alvo*, constatou-se que o processo de GCS elaborado apóia as abordagens experimentadas para obtenção da equivalência entre os sistemas legado e alvo. Dessa forma, caso sejam solicitadas, as alterações podem ser realizadas, sucessivas vezes, no sistema alvo a fim de se obter a aderência do sistema alvo aos requisitos do sistema legado (Cagnin, 2005). Caso o sistema seja novamente gerado, as alterações não são perdidas e podem ser incorporadas às novas versões.

Alterações podem ser também realizadas no mapeamento XML, como proposto por Freitas (2006). O processo de GCS apóia a preservação (*trunk*) tanto do GAwCRe quanto da linguagem de padrões SiGLi e quaisquer adaptações para domínios conexos podem ser criadas utilizando ramificações (*branch*), o que permite a exploração da variabilidade do gerador.

Quatro cenários são identificados quando se pretende utilizar o gerador GAwCRe para a realizar a reengenharia de sistemas legados. Na Figura 5.13 são exibidos os cenários identificados e que são comentados a seguir:

1. O sistema legado pertence exatamente ao mesmo domínio do gerador;

2. O sistema legado pertence a domínio conexo ao gerador, e as funções que não são apoiadas pela linguagem de padrões precisam ser implementadas;

3. O sistema legado pertence ao domínio do gerador, entretanto sua funcionalidade representa um subconjunto da funcionalidade da linguagem de padrões. Portanto o sistema alvo é gerado com funções a mais que o desejado. O sistema Fisiosoftware se enquadra nesse cenário;

4. Este quarto cenário combina características dos cenários 2 e 3. O sistema legado pertence a domínio conexo ao do gerador e possui funções a mais que devem ser implementadas, entretanto as funções pertencentes ao domínio do gerador são na verdade um subconjunto dessas. Portanto o sistema alvo é gerado com funções a mais que podem ser eliminadas ou aceitas como evolução do sistema. E também funções a menos que deverão ser implementadas. O sistema *Psychologist Software* enquadra-se nesse cenário, bem como os sistemas que passaram

por reengenharia nos estudos de caso realizados por Freitas (2006) e Ferreira (2007).

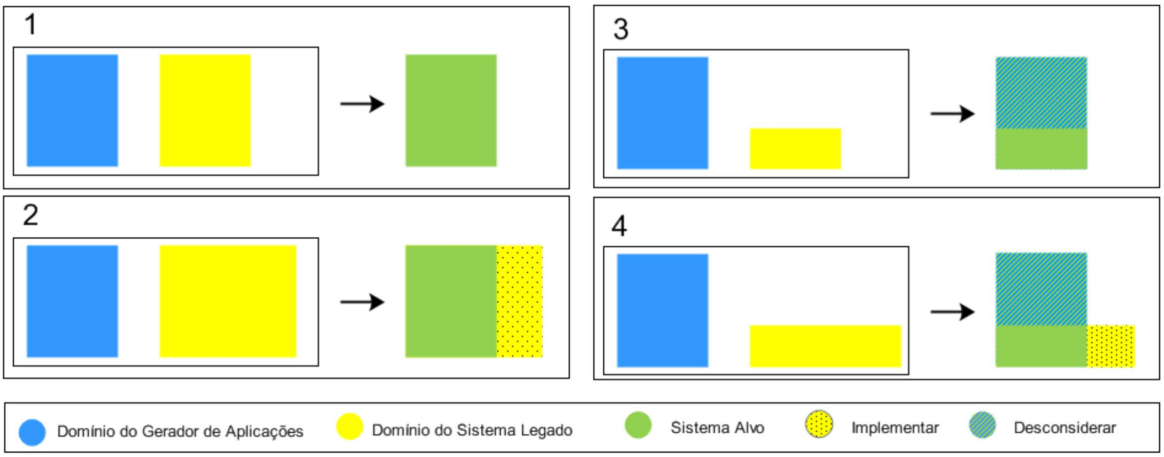

Figura 5.13: Domínio do gerador GAwCRe contrastado ao domínio de sistemas legados hipotéticos

Inferiu-se portanto, a necessidade de que estudos mais aprofundados sejam feitos para verificar a possibilidade de expansão do domínio da SiGCLi ou realizar a reestruturação de seus padrões, pois, somente alterando o mapeamento XML não é possível a reengenharia completa desses sistemas. O principal problema encontrado reside na incompatibilidade de classes, de atributos e de métodos existentes na linguagem de padrões SiGCli e os sistemas legados utilizados.

CAPÍTULO
6

Processo Ágil de Reengenharia com Geradores de Aplicações e GCS

6.1 Considerações Iniciais

A reengenharia de software é uma tarefa, inerentemente, suscetível a riscos, por esse motivo, o conhecimento prévio desses riscos pode contribuir para analisá-los, gerenciá-los e controlá-los de modo mais eficaz. Entre os riscos comentados por Rosenberg (1996) estão: Não cumprimento dos prazos e custos previstos; escolha inadequada de subsistemas submetidos à reengenharia; alto custo envolvido na reengenharia; alto custo da documentação; extenso volume de documentação; gerência de configuração inadequada; baixa capacitação das pessoas envolvidas na reengenharia; ausência de garantia da qualidade do sistema resultante; performance inadequada do sistema alvo; ausência de um programa de métricas; emprego de tecnologia inadequada para a realização da reengenharia; algumas funcionalidades do sistema tornam-se obsoletas antes do término da reengenharia; insuficiência da reengenharia; dificuldades na recuperação do projeto e dos requisitos a partir do código fonte; perda das regras do negócio embutidas no código fonte do legado e dificuldades para realizar a migração dos dados do legado.

O arcabouço ARA (Cagnin, 2005) é um exemplo de abordagem de reengenharia ágil, que apresenta um conjunto de recursos que colaboram para que a reengenharia seja realizada com redução de tempo, de custos e de alguns dos riscos identificados por Rosenberg (1996). Originalmente ARA foi utilizado com o processo PARFAIT (Cagnin, 2005) que utiliza frameworks para a instanciação de sistemas. A neces-

sidade de controle de versões, no contexto de frameworks é ressaltada por Cagnin (2005) tanto para controlar as versões do framework quanto as das aplicações criadas por meio de sua instanciação. Desse modo, foi elaborada a ferramenta GREN-WizardVersionControl (Cagnin, 2005) que realiza o controle de versões das aplicações criadas pelo PARFAIT usando o framework GREN (Braga, 2002).

A incorporação da GCS ao processo de reengenharia ágil utilizando geradores de aplicação descrito no Capítulo 5, possibilitou a recuperação parcial de versões intermediárias do software produzido pelo gerador. O estudo de caso realizado possibilitou o aprimoramento do processo e a identificação dos diferentes produtos produzidos pelo gerador quando a reengenharia é realizada em sistemas pertencentes a domínios conexos ao que o gerador foi criado.Na próxima Seção as fases e atividades do Processo Ágil de Reengenharia Utilizando Geradores de Aplicações e GCS são apresentadas.

Este Capítulo abstrai o processo ágil de reengenharia para geradores de aplicações com GCS e encontra-se organizado do seguinte modo. Na Seção 6.2 é definido um modelo para a apresentação das atividades que compõem as fases do Processo Ágil de Reengenharia Utilizando Geradores de Aplicações e GCS, na Seção 6.3 essas atividades são apresentadas de modo que possam ser usadas com qualquer gerador e na Seção 6.4 são feitas as considerações finais.

6.2 Modelo Utilizado para Descrição das Atividades

A fases de Concepção, Elaboração, Construção e Transição que compõem o Processo Ágil de Reengenharia com Geradores de Aplicações e GCS foram herdadas do processo PARFAIT (Cagnin, 2005) e são utilizadas neste trabalho no contexto de reengenharia. Entretanto para acomodar as tarefas de GCS, algumas atividades tiveram que ser adaptadas. Cada atividade descreve o contexto no qual os problemas abordados ocorrem e quais as soluções propostas pela atividade para elucidá-los. O formato da apresentação consiste nos seguintes elementos:

Fase: nome da fase em que a atividade deve ser executada.

Nome: denominação da atividade, deve ser clara e concisa.

Responsável: descreve o(s) encarregado(s) da realização da atividade. (Por exemplo: analista de sistemas, gerente de configuração e desenvolvedor)

Objetivo descreve o propósito da realização de atividade..

Atividades relacionadas: documenta as atividades relacionadas à qual está sendo descrita. Essa seção apresenta as atividades que podem ser aplicadas concomitantemente, anteriormente ou posteriormente à aplicação da atividade descrita.

Pré-condições: descreve condições necessárias para aplicação da atividade.

Pós-condições: descreve os resultados obtidos por meio da aplicação da atividade, bem como o conjunto de ferramentas disponíveis, visto que a realização de algumas atividades implica na introdução e utilização de determinadas ferramentas de apoio.

Fluxo de atividades: descreve a seqüência de passos necessários para a execução da atividade.

6.3 Atividades do Processo Ágil de Reengenharia com Geradores de Aplicações

6.3.1 Atividade: *Familiarizar-se com o Domínio de Gerador de Aplicações*

Fase: Concepção.

Nome: *Familiarizar-se com o Domínio de Gerador de Aplicações*.

Responsável: Analista de sistemas

Objetivo: Analisar e entender o domínio ao qual o gerador pertence. Verificar a funcionalidade implementada pelo gerador e exercitar tal funcionalidade por meio da criação de sistemas alvo hipotéticos.

Motivação: Obter visão geral do escopo do domínio ao qual o gerador está inserido, e obter informações de alto nível relacionadas à complexidade do domínio para prever riscos relacionados. Essa análise também fornece noção de tempo e de custo que serão necessários para a realização da reengenharia.

Atividades relacionadas: nenhuma.

Pré-condições:

- O gerador deve estar instalado e corretamente configurado;
- A documentação do gerador e de seu domínio, caso esteja disponível deve ser avaliada;
- O ambiente de GCS deve ter sido previamente estabelecido, isso implica que um plano de GCS deve ter sido elaborado e o conjunto de ferramentas de apoio necessárias para automatização do mesmo deve estar devidamente instalado e configurado.

Pós-condições:

- Gerador instalado e configurado;
- Documentação do gerador e de seu domínio de atuação disponibilizada e/ou atualizada;
- Ambiente de GCS estabelecido;
- Conjunto de ferramentas de apoio necessárias para às atividades de GCS instalado e configurado.

Fluxo de atividades:

1. O analista de sistemas consulta documentos relacionados à geração de aplicações utilizando o gerador, quando disponíveis.
2. O analista de sistemas gera aplicações com base no conhecimento anteriormente obtido.

6.3.2 Atividade: *Observar o Domínio do Sistema Legado*

Fase: Concepção.

Nome: *Observar o Domínio do Sistema Legado*.

Responsável: Analista de Sistemas, Gerente de Configuração.

Objetivo: Analisar e entender as características do sistema legado em relação ao domínio do gerador, verificando se ele pertence ao mesmo domínio, a um domínio conexo ao que o gerador foi desenvolvido ou desconexo. Contrastando-se o domínio do sistema legado com o domínio do gerador é possível avaliar sua aplicabilidade para realização da reengenharia. Dessa forma, caso o gerador pertença a um domínio desconexo não é possível utilizá-lo para a reengenharia do sistema legado. Essa atividade também identifica se o sistema legado possui documentação associada.

Atividades relacionadas: *Familiarizar-se com o Domínio de Gerador de Aplicações*.

Pré-condições:

- A documentação relacionada ao sistema legado, se disponível, deve ser avaliada;
- O sistema legado deve estar instalado e configurado;
- Um repositório para o projeto deve estar disponível na ferramenta de controle de versões.

Pós-condições:

- Sistema legado instalado e configurado;
- Documentação do sistema legado disponibilizada;
- Documento de aceitação do gerador justificando se o projeto de reengenharia deve ou não continuar.

Fluxo de atividades:

1. O gerente de configuração cria um repositório para o projeto;
2. O analista de sistemas executa o sistema legado e registra as funções identificadas.
3. O analista de sistemas persiste as informações obtidas sob o controle da GCS a fim de facilitar a rastreabilidade, o gerenciamento e a evolução dessas informações.
4. O analista de sistemas compara as funções identificadas com as funções possíveis de serem implementadas pelo gerador;

6.3.3 Atividade: *Desenvolver o Diagrama de Casos de Uso e Elaborar os Casos de Teste*

Fase: Elaboração.

Nome: *Desenvolver o Diagrama de Casos de Uso e Elaborar os Casos de Teste*.

Responsável: Analista de sistemas, Usuários.

Objetivo: Elaborar um diagrama de casos de uso contendo as funções do sistema legado identificadas durante a aplicação da atividade *Observar o Domínio do Sistema Legado*. O desenvolvimento do diagrama de casos de uso apóia a elaboração dos casos de teste que serão utilizados para o entendimento dos requisitos do sistema legado. Os casos de teste são elaborados e deve se estabelecer a ordem de prioridade com que as funções identificadas serão geradas. Os requisitos são examinados pelo analista de sistemas e as funções são classificadas de acordo com a prioridade atribuída pelo analista e/ou usuários.

Atividades relacionadas: *Observar o Domínio do Sistema Legado*.

Pré-condições:

- Descrição em alto nível das funções do sistema legado.

Pós-condições:

- Diagrama de casos de uso das funções do sistema legado;
- Documentação dos casos de teste;
- Lista das funções identificadas e priorizadas para geração.

Fluxo de atividades:

1. O analista de sistemas executa o sistema legado para descobrir os cursos de execução normais e alternativos;
2. O analista de sistemas elabora o diagrama de casos de usos;
3. O analista de sistemas elabora os casos de teste de cada curso de execução;
4. O analista de sistemas valida com os usuários do sistema legado cada caso de uso criado;
5. Todos os documentos produzidos são enviados ao repositório do projeto pelo analista de sistema.

6.3.4 Atividade: *Criar o Sistema Alvo*

Fase: Construção.

Nome: *Criar o Sistema Alvo*.

Responsável: Programador.

Objetivo: Criar uma versão do sistema alvo utilizando o gerador. A criação de uma versão operacional do sistema alvo prematuramente colabora para minimizar o risco de elicitação de objetos incompleta e incorreta.

Atividades relacionadas: *Desenvolver o Diagrama de Casos de Uso* e *Elaborar os Casos de Teste*.

Pré-condições:

- A documentação dos casos de teste e o diagrama de casos de uso devem estar disponíveis para consulta;
- A relação de funções priorizadas para geração do sistema alvo deve estar disponível.

Pós-condições:

- Uma versão inicial do sistema alvo é criada e persistida no repositório da ferramenta de controle de versões.

Fluxo de atividades:

1. O programador utiliza o gerador de aplicações para gerar o sistema alvo;
2. O programador persiste a versão inicial do sistema alvo no repositório do projeto.

6.3.5 Atividade: *Executar os Casos de Teste no Sistema Alvo*

Fase: Construção.

Nome: *Executar os Casos de Teste no Sistema Alvo*.

Responsável: Testador, Usuários.

Objetivo: Descobrir se regras de negócio e requisitos do sistema legado não foram gerados e não estão presentes no sistema alvo. Identificar se regras de negócio e requisitos disponíveis no gerador, e que não fazem parte dos requisitos do sistema legado foram gerados e estão presentes no sistema alvo.

Atividades relacionadas: *Desenvolver o Diagrama de Casos de Uso* e *Elaborar os Casos de Teste*.

Pré-condições:

- Documentação dos casos de teste.

Pós-condições:

- Documentação de funções desnecessárias presentes no sistema alvo e também de funções priorizadas para geração e que não foram geradas.
- Documentação gerada persistida no repositório do controle de versões.

Fluxo de atividades:

1. O testador de software deve adequar os casos de teste para serem aplicados no sistema alvo;
2. O testador de software executa no sistema alvo os casos de teste documentados;
3. O testador de software compara os resultados obtidos com os resultados esperados.

6.3.6 Atividade: *Adaptar o Sistema Alvo*

Fase: Construção.

Nome: *Adaptar o Sistema Alvo*.

Responsável: Programadores, Analista de Sistemas.

Objetivo: Obter a cada iteração uma versão do sistema alvo mais consistente com os requisitos do sistema legado. Regras de negócio e requisitos que não são cobertos pelo gerador, ou que foram gerados e não são necessários são exemplos de adaptações que devem ser realizadas, para que haja equivalência entre os requisitos do sistema legado e os do sistema alvo. Sugestões recolhidas dos usuários, quanto a modificações dos requisitos, também devem ser analisadas e consideradas como possíveis adaptações.

Atividades relacionadas: *Desenvolver o Diagrama de Casos de Uso*, *Elaborar os Casos de Teste*, *Criar o Sistema Alvo*, *Executar os Casos de Teste no Sistema Alvo* e *Documentação dos casos de teste*.

Pré-condições:

- Versão do sistema alvo deve ter sido criada;
- Solicitações de mudanças devem ter sido registradas na ferramenta de controle de modificações;
- Documentação dos resultados dos casos de testes devem estar disponíveis;
- Lista das funções identificadas e priorizadas para geração deve estar disponível.

Pós-condições:

- Nova versão do sistema alvo adaptada é obtida e deve ser armazenada no repositório da ferramenta de controle de versões;
- Registro da solicitação de mudança devem ser atualizados;
- Artefatos produzidos e atualizados são persistidos no repositório da ferramenta de controle de versões.

Fluxo de atividades:

1. Os programadores planejam as adaptações e as registram na ferramenta de controle de mudanças;

2. Os envolvidos no projeto aprovam, rejeitam ou sugerem alterações no planejamento das adaptações;

3. Os programadores implementam as modificações, persistem os artefatos modificados no repositório do sistema de controle de versões e atualizam o registro das solicitações de mudanças;

6.3.7 Atividade: *Converter a Base de Dados*

Fase: Transição.

Nome: *Converter a Base de Dados*.

Responsável: Administrador de banco de dados.

Objetivo: Migrar a base de dados do sistema legado para a do sistema alvo, se o sistema legado encontra-se em produção, para preservação dos dados existentes.

Atividades relacionadas: *Criar o Sistema Alvo*.

Pré-condições:

- Ferramenta de conversão de dados deve estar instalada e configurada;
- É necessário ter permissão de acesso à base de dados do sistema legado em produção;
- A ferramenta de SGBD do sistema alvo deve estar instalada e configurada.

Pós-condições:

- Base de dados do sistema legado deve ter sido transferida para a base de dados do sistema alvo;
- A base de dados do sistema legado em produção é preservada.

Fluxo de atividades:

- O administrador de banco de dados determina a correspondência entre a base de dados do sistema legado com a do sistema alvo;
- O administrador de banco de dados migra os dados do sistema legado para o sistema alvo.

6.3.8 Atividade: *Testar o Sistema Alvo*

Fase: Transição.

Nome: *Testar o Sistema Alvo*.

Responsável: Testador de software.

Objetivo: Executar os casos de teste documentados a fim de se avaliar a maturidade da versão final do sistema alvo produzido para liberação do sistema para produção.

Atividades relacionadas: *Desenvolver o Diagrama de Casos de Uso*, *Elaborar os Casos de Teste*, *Criar o Sistema Alvo*, *Executar os Casos de Teste no Sistema Alvo*, *Documentação dos casos de teste* e *Converter a Base de Dados*.

Pré-condições:

- Versão do sistema alvo deve estar disponível para testes;
- Documentação dos casos de teste deve estar disponível;
- A base de dados do sistema legado deve ter sido migrada para a do sistema alvo.

Pós-condições:

- Versão final do sistema alvo liberada para produção.

Fluxo de atividades:

1. O testador de software executa novamente os casos de teste no sistema alvo;
2. O testador de software verifica o resultado obtido contrastando-o com os resultados esperados;
3. Caso os resultados não sejam satisfatórios, o testador de software comunica ao analista de sistemas e aos programadores que novas adaptações devem ser realizadas.

6.3.9 Atividade: *Desenvolver o Diagrama de Classes do Sistema Alvo*

Fase: Transição.

Nome: *Desenvolver o Diagrama de Classes do Sistema Alvo*.

Responsável: Analista de sistemas.

Objetivo: Documentação do sistema gerado.

Atividades relacionadas: nenhuma.

Pré-condições:

- Ferramenta de engenharia reversa deve estar disponível para análise do código fonte da aplicação gerada. Caso contrário, uma ferramenta de modelagem deve estar disponibilizada para elaboração do modelo de classes.

Pós-condições:

- Modelo de classes do sistema alvo deve ter sido elaborado.

Fluxo de atividades:

1. O analista de sistemas obtém o modelo de classes da versão final do sistema alvo por meio da ferramenta de engenharia reversa, que analisa o código fonte do sistema alvo e gera o modelo de classes.

   (a) Caso nenhuma esteja disponível, o modelo deverá ser elaborado utilizando ferramenta de modelagem.

2. O analista de sistemas deve enviar ao repositório do sistema de controle de versões os artefatos produzidos e os modificados.

6.4 Considerações Finais

Este capítulo apresentou as atividades que devem ser realizadas quando o Processo Ágil de Reengenharia com Geradores de Aplicações utiliza GCS. Esse processo apóia a utilização do ARA com geradores de aplicações, contribuindo para que a reengenharia de sistemas legados ocorra em ambiente controlado, no qual todas as etapas significantes são documentadas e rastreadas e passíveis de serem reproduzidas.

O processo modificado possibilita, por exemplo, que o gerador disponível e também o código fonte do sistema alvo sejam modificados por meio dos passos propostos pela atividade *Adaptar o Sistema Alvo*. Dessa forma, a variabilidade de produtos que podem ser criados, a partir do gerador considerado pode ser explorada, sem prejuízo da LPS que está inserida nesse gerador.

Ao fim de cada atividade é recomendado que os artefatos criados e os modificados sejam enviados ao repositório do sistema de controle de versões.

CAPÍTULO
7

Conclusão

7.1 Visão Geral

Arcabouço de Reengenharia Ágil – ARA faz uso de processos, ferramentas, abordagens de reúso para apoiar a migração de sistemas legados procedimentais para o paradigma orientado a objetos. A automatização do processo de reengenharia de sistemas legados pode ser aprimorada com o uso de ferramentas de apoio que possibilitam a reutilização ou geração automatizada de código fonte, tais como frameworks e geradores de aplicações, respectivamente. O desenvolvimento, utilização e manutenção desses dois tipos de técnicas estão inseridos no contexto de Linha de Produtos de Software (LPS). É recomendado que o os recursos que implementam a funcionalidade as variabilidades existentes na LPS estejam sob o controle de atividades de Gerência de Configuração de Software (GCS). Esse controle é necessário para possibilitar a evolução controlada da LPS e também dos artefatos desenvolvidos, evitando que alterações inapropriadas aconteçam (Staples, 2004).

A automatização do processo de reengenharia de sistemas legados pode ser facilitado por meio do uso de geradores de aplicações, visto que são capazes de gerar sistemas de software a partir de especificações em alto nível. Entre as vantagens pode-se citar a possibilidade de obtenção de produtos com qualidade, todavia produzidos em espaço de tempo menor se comparado ao desenvolvimento de software tradicional; a linguagem de padrões, em nível de análise, utilizada com o gerador pode auxiliar no conhecimento do domínio do sistema legado.

A utilização do Processo Ágil de Reengenharia com Geradores de Aplicações (Freitas, 2006) sem atividades de GCS comprometem o uso do gerador com aumento de retra-

balho, uma vez que não há controle da composição dos artefatos gerados.

As contribuições deste trabalho estão apresentadas na Seção 7.2. Algumas limitações quanto a restrições consideradas durante o desenvolvimento da proposta são comentados na Seção 7.3 e na Seção 7.4 são listados trabalhos que podem dar continuidade a este.

7.2 Contribuições

O Processo Ágil de Reengenharia com Geradores de Aplicações e GCS é um aprimoramento em relação ao Processo Ágil de Reengenharia Utilizando Geradores de Aplicações proposto por Freitas (2006). Esse processo permite que a abordagem ARA utilize geradores de aplicações, ao invés de frameworks, como ferramenta de apoio na geração automatizada de aplicações. O diferencial nessa nova versão do processo é a utilização de práticas de GCS.

Para viabilizar a implantação das atividades de GCS, um Plano de Gerência de Configuração de Software foi elaborado com base na norma IEEE 828-2005. O plano elaborado é utilizado em associação ao Processo Ágil de Reengenharia com Geradores de Aplicações sem comprometer sua agilidade. O plano elaborado enfoca somente atividades estratégicas de GCS, tais como controle de mudanças; identificação de itens de configuração e técnicas de armazenamento; autorização de mudanças; gerenciamento de construções e relatório de status. Para verificar essa proposta o gerador de aplicações GAwCRe (Pazin, 2004) foi utilizado. As manutenções nele realizadas contribuíram para aprimorá-lo, possibilitando sua evolução e manutenções futuras.

O manual de usuário do gerador GAwCRe e um manual de instalação para o gerador e seus artefatos foram elaborados e também foram documentados todos os passos necessários para a implantação do ambiente de GCS para dar suporte ao Processo Ágil de Reengenharia com Geradores de Aplicações e GCS. Além disso, automatizou-se a construção da versão `RB_03` do gerador, utilizando a ferramenta Ant (Apache Ant, 2008), sendo elaborados *scripts*. Esses *scripts*, armazenados no repositório do projeto, podem ser adaptados para serem empregados no contexto de outras versões do GAwCRe ou de outro gerador.

O processo de compilação das classes geradas pelo GAwCRe e a criação de tabelas também foi automatizado, facilitando a instalação das aplicações geradas e minimizando a ocorrência de erros.

A abordagem de GCS definida, permite também a exploração da variabilidade do GAwCRe, à medida em que as várias versões armazenadas podem ser utilizadas para a geração de aplicações diferentes.

7.3 Limitações do Trabalho Realizado

Algumas limitações foram encontradas durante a realização deste trabalho. Com relação às manutenções pretendidas, não foi possível realizar a separação entre o código de apresentação (CSS) e o código de estruturação (HTML) presente nos templates do gerador e que são utilizados na criação das páginas JSP. Essas alterações foram dificultadas, pois o gerador não contém testes que podem atuar como testes de regressão, executados após a realização das alterações.

Com relação ao domínio do gerador, somente foram testadas soluções como inclusão e alterações de atributos e métodos. Não foi possível expandir o domínio da SiGCli para outras clínicas médicas, pois da maneira como os seus padrões estão organizados não são permitidas muitas alterações sem comprometer a linguagem (Borges et al., 2008).

Outra limitação está relacionada a não automatização de todas as tarefas inicialmente planejadas, que ficou restrita à versão utilizada neste trabalho. O plano de GCS elaborado foi testado somente com o Processo Ágil de Reengenharia com Geradores de Aplicações e GCS e com o gerador de aplicações GAwCRe, dessa forma, estudos adicionais devem ser realizados a fim de avaliar a sua aplicabilidade e flexibilidade. Para tal, deve-se realizar a reengenharia de outros sistemas legados, além dos relatados neste trabalho, utilizando o processo proposto, o que possibilitará o refinamento, por meio da introdução de novas práticas ou técnicas, bem como a evolução do processo.

7.4 Trabalhos Futuros

A seguir são listados alguns trabalhos que podem dar continuidade ao realizado nesta dissertação:

- Realizar a reengenharia de sistemas legados utilizando outros geradores de aplicações, a fim de verificar a consistência e a abrangência do processo elaborado;
- Somente com as alterações realizadas no mapeamento XML não foi possível expandir o domínio da SiGCli para domínios conexos, pois, da maneira como os seus padrões estão organizados não são permitidas muitas alterações. Para isso é necessário o estudo dos domínios e a reestruturação da SiGCli, com a inclusão de novos padrões e possível alteração nos já existentes;
- A atual interface gráfica com o usuário do gerador possibilita que poucas customizações sejam efetuadas nos artefatos produzidos. Uma possível alteração na apresentação dos padrões poderia ser feita, permitindo um maior refinamento na escolha dos atributos e métodos a serem implementados;

- No código fonte do gerador GAwCRe foram encontrados muitos métodos extensos e código fonte duplicado, que dificulta a sua compreensão e também a realização de alterações. Assim, atividades de manutenção preventiva (refatorações) podem ser realizadas;
- Separação do código de apresentação (CSS) do código de estruturação (HTML), o que possibilitaria a customização da interface gráfica com o usuário das aplicações geradas pelo GAwCRe;

Apêndices

APÊNDICE

A

Plano de Gerência de Configuração

DATA	ITEM MODIFICADO	SUMÁRIO DAS MUDANÇAS
15/03/2008	Versão Inicial	Em elaboração

A.1 Plano de Gerência de Configuração

A.1.1 Introdução

O GAwCRe é um gerador de aplicações para a web desenvolvido com base em Linhas de Produto de Software no domínio de clínicas de reabilitação física (Fisioterapia, Terapia Ocupacional e Educação Física). O GAwCRe permite a instanciação de diferentes aplicações, cabendo ao desenvolvedor escolher os padrões que irão fazer parte de sua aplicação a partir de uma interface gráfica. A instanciação é feita a partir de uma Linguagem de Modelagem de Aplicação (LMA) definida com base na linguagem de padrões SiGCli.

A.1.2 Planejamento da Gerência de Configuração de Software

Este plano visa garantir a integridade dos itens de configuração relacionados à manutenção e evolução do gerador de aplicações GAwCRe e de seus artefatos. Para tal foram definidas algumas abordagens para apoiar a utilização do gerador, tanto para a geração de aplicações no domínio da SiGLi, quanto para a reengenharia de sistemas legados pertencentes a domínios conexos ao da SiGLi.

O plano foi definido para auxiliar:

- O gerenciamento do estado dos ICs;
- O controle das solicitações de mudanças;
- O rastreamento das modificações nos ICs ao longo do tempo.

A.1.3 Referências

DOCUMENTO	FONTE
IEEE Guide to Software Configuration Management ANSI/IEEE std1042-1987	`http://ieeexplore.ieee.org/Xplore/-guesthome.jsp`
IEEE Std 828-2005	`http://ieeexplore.ieee.org/-Xplore/login.jsp?url=/iel5/10048/-32241/01502775.pdf?arnumber=1502775`
Manual do Gerador GAwCRe	a ser disponibilizado (Borges e Penteado, 2008a)
Manual de Instalação e Configuração do Ambiente de GCS	a ser disponibilizado (Borges e Penteado, 2008c)

A.1.4 Responsáveis

A.1.5 Permissões

O portal Assembla hospeda o repositório do projeto, nele é possível definir dois tipos de usuários: *owner*, que possui todos os privilégios de administrador e *member*, que possui somente os privilégios definidos por um administrador (*owner*). É preciso controle rigoroso do acesso ao repositório para garantir a integridade dos artefatos armazenados. A política de acesso aos diretórios no repositório é de responsabilidade do Gerente de Configuração.

A.2 Itens de Configuração

- Documentação relativa aos requisitos, especificações, etc;

PAPEL	Gerente de Configuração
ATIVIDADE A SER DESENVOLVIDA	• Estruturar o ambiente de desenvolvimento; • Desenvolver a estrutura do repositório; • Identificar e estabelecer a configuração; • Garantir o registro de todas as alterações nos ICs.
NOME(S) DO(S) RESPONSÁVEL(EIS)	–

PAPEL	Gerente do Sistema
ATIVIDADE A SER DESENVOLVIDA	• Responsável pela autorização das modificações nos ICs; • Garantir a utilização das ferramentas de apoio.
NOME(S) DO(S) RESPONSÁVEL(EIS)	–

- Modelos de análise e projeto;
- Arquivos de configuração;
- Manuais;
- Bibliotecas, componentes, plugins, etc;
- Código-fonte;
- Imagens;
- Outros artefatos, desde que identificados com o propósito da criação e do armazenamento.

A.3 Estrutura do Repositório

Na raiz encontram-se três diretórios que permitem a organização do ambiente de desenvolvimento. Utilizou-se o modelo *trunk*, *tag*, *branch* como uma forma de padronização, por ser um modelo difundido de estruturação facilitando a utlização.

A.3.1 Organização do Repositório

Diretório: *trunk*.

PAPEL	Analistas e Desenvolvedores
ATIVIDADE A SER DESENVOLVIDA	• Analisam o impacto das modificações solicitadas; • Propõem soluções para as solicitações e as implementam.
NOME(S) DO(S) RESPONSÁVEL(EIS)	–

Descrição: Nele está armazenado a versão original do GAwCRe.

Observações: Inicialmente a estrutura de pastas de diretórios definida por Pazin (2004) foi mantida. Entretanto ao elaborar a versão `RB_03` e adotar o modelo jakarta, decidiu-se por reestruturar todas as outras ramificações, evoluindo seu formato e adequando-o ao utilizado na versão `RB_03`. Desta forma se dará o aproveitamento dos scripts de automação já escritos, que assim podem ser reutilizados mais facilmente visto que doravante, todas as versões apresentam a mesma estrutura organizacional.

Diretório: *branch*.

Descrição: São armazenadas as versões produzidas do gerador.

Observações: –

Diretório: *tag*.

Descrição: (a ser definido).

Observações: A utilização de *tags* não foi explorada no trabalho que deu origem a este plano de GCS.

O repositório definido segue o modelo jakarta. O modelo não representa nenhum *layout* específico utilizado por IDEs. Os ICs são organizados de acordo com sua funcionalidade. Por este motivo a construção de *builds* e *releases* manuais torna-se complexa, devendo ser criados *scripts* de automação para executá-las.

Arquivo `glossário.txt`: Arquivo contento as definições e termos utilizados, visando nivelar o vocabulário dos envolvidos a cerca desses termos.

Arquivo `leiame.txt`: Arquivo com instruções a respeito do conteúdo do diretório, onde obter maiores informações, recomendações, etc.

Diretório `/<nome_da_versão>`: A versão número zero do gerador chama-se GAwCRe, as demais seguem um padrão de numeração.

Diretório `/analise_projeto`: Todos os documentos gerados por atividades de análise e projeto.

Diretório `/banco_dados`: *Scripts* de banco de dados, documentos de modelagem ER, etc.

Diretório `/bibliotecas`: Arquivos que não foram produzidos pelo projeto, mas são utilizados para apoiar o gerador. *Plugins*, componentes, bibliotecas (*library*), devem ser armazenados aqui.

Diretório `/código_fonte`: Código-fonte do gerador, arquivos diversos que são distribuídos com as aplicações geradas, o mapeamento da linguagem de padrões em formato de documento XML.

Diretório `/documentação`: Anotações genéricas, documento de cronograma, *brainstoming*, etc

Diretório `/manuais`: Documentos que explicam a instalação das ferramentas de apoio, do gerador, artefatos gerados, configuração do gerador, etc.

Diretório `/worktemp`: Diterório para auxiliar a criação de *builds*, é utilizado por alguns *scripts* de instalação dos artefatos produzidos pelo GAwCRe.

Diretório `/requisitos`: Reservado para a documentação dos requisitos da evolução do gerador, mas principalmente para a reengenharia de sistemas legados utilizando o GAwCRe.

A.4 Gerência de Versões – Formato do número das versões dos itens de configuração

Deve ser definido pelo gerente de configuração responsável pelo projeto.

Referências Bibliográficas

Ahern, D. M., Clouse, A., e Turner, R. (2008). *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*. Addison-Wesley Professional.

Alexander, C. (1977). *A Pattern Language: Towns, Building and Construction*. Oxford University Press.

Ambler, S. W. e Jeffries, R. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. Wiley.

Apache Ant (2008). The Apache Ant Project. Disponível em: *http://ant.apache.org/*. (Acessado 11 de Dezembro de 2007).

Apache Jakarta (2008). The Apache Jakarta Project. Disponível em: *http://jakarta.apache.org/site/dirlayout.html*. (Acessado 30 de Abril de 2008).

Apache Tomcat (2008). Apache Tomcat. Disponível em: *http://tomcat.apache.org/*. (Acessado 30 de Janeiro de 2008).

Apache Xerces (2007). The Apache XML Project. Disponível em: *http://xml.apache.org/*. (Acessado 1 de Maio de 2008).

Asklund, U. e Bendix, L. (2002). A study of configuration management in open source software projects. *IEE Software*, 149:40–46.

Assembla (2008). Accelerating Software Development. Disponível em: *http://www.assembla.com/*. (Acessado 21 de Junho de 2008).

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Biomanager (2008a). FisioSoftware. Disponível em: *http://www.biomanager.com.br/FisioSoftware/index.htm*. (Acessado em 09 de março de 2008).

Biomanager (2008b). Psychologist Software. Disponível em: *http://www.biomanager.com.br/PsicoSoft/*. (Acessado em 09 de março de 2008).

Borges, S. S. (2007). *Um Estudo de Qualidade na Manutenção de Gerador de Aplicações*. Documento de qualificação, Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP.

Borges, S. S., Ferreira, T. T., e Penteado, R. A. (2008). Manutenção e Evolução de um Gerador de Aplicações Desenvolvido com Linha de Produtos de Software. *Workshop de Manutenção de Software Moderna*.

Borges, S. S. e Penteado, R. A. D. (2008a). Manual de Instalação e Configuração do Ambiente de GCS.

Borges, S. S. e Penteado, R. A. D. (2008b). Manual de Instalação e Configuração do Gerador GAwCRe e de Sistemas Produzidos pelo Gerador.

Borges, S. S. e Penteado, R. A. D. (2008c). Visão Geral das Atividades de Manutenção Realizadas.

Braga, R. T. V. (2002). *Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico*. Tese de doutorado, ICMC/USP, São Carlos - SP.

Braga, R. T. V., Germano, F. S. R., e Masiero, P. C. (1999). A Pattern Language for Business Resource Management. *Proceedings of the Annual Conference on Pattern Languages of Programs*, 6:1–33.

Braga, R. T. V. e Masiero, P. (2002). A Process for Framework Construction Based on a Pattern Language. *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*.

Cagnin, M., Maldonado, J., e Penteado, R. (2003). Parfait: towards a framework-based agile reengineering process. *Agile Development Conference, 2003. ADC 2003. Proceedings of the*, pages 22–31.

Cagnin, M. I. (2005). *PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks*. Tese de doutorado, ICMC/USP, São Carlos - SP.

Canfora, G. e Penta, M. D. (2007). New frontiers of reverse engineering. *Future of Software Engineering, 2007. FOSE '07*, pages 326–341.

Clark, M. (2004). *Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Apps*. The Pragmatic Programmers.

Cleaveland, J. C. (2001). *Program generators with Java and XML*. Prentice-Hall.

CollabNet (2008a). CollabNet: Where Subversion Meets the Enterprise. Disponível em: *http://www.collab.net/products/subversion/*. (Acessado 2 de Maio de 2008).

CollabNet (2008b). The coolest Interface to (Sub)Version Control. Disponível em: *http://tortoisesvn.tigris.org/*. (Acessado 2 de Maio de 2008).

Collins-Sussman, B., Fitzpatrick, B. W., e Pilato, C. M. (2007). *Version Control with Subversion*. O'Reilly Media, Inc.

Conradi, R. e Westfechtel, B. (1997). Towards a uniform version model for software configuration management. In *Software Configuration Management*, Lecture Notes in Computer Science, pages 1–17. Springer-Verlag.

Coplien, J. (1996). *Software Patterns*. Sigs Books.

Eclipse (2007). The Eclipse Foundation. Disponível em: *http://www.eclipse.org/*. (Acessado 20 de Fevereiro de 2007).

Eisenecker, U. W. e Czarnecki, K. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

Estublier, J. (2000). Software configuration management: a roadmap. In *International Conference on Software Engineering ICSE - Future of SE Track*, pages 279–289.

Estublier, J., Leblang, D., van der Hoek, A., Conradi, R., Clemm, G., Tichy, W., e Wiborg-Weber, D. (2005). Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol.*, 14(4):383–430.

Ferreira, T. T. (2007). *Avaliação de um Processo de Manutenção Usando Gerador de Aplicações*. Projeto de iniciação científica, Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP.

Flanagan, D. (2002). *JavaScript Pocket Reference*. O'Reilly & Associates, Inc.

Fowler, M. (1996). *Analysis Patterns: Reusable Object Models*. Addison-Wesley.

Fowler, M. (2003). Patterns. *Software, IEEE*, 20:2–3.

Fowler, M. (2005). UML E*ssencial:* U*m breve guia para a linguagem-padrão de modelagem de objetos*. Bookman. 3° Edição.

Fowler, M., Beck, K., Brant, J., Opdyke, W., e Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Franca, L. P. A. e Staa, A. V. (2001). Geradores de Artefatos: Implementação e Instanciação de Frameworks. *Simpósio Brasileiro de Engenharia de Software (SBES)*, pages 302–315.

Franca, L. P. A. e Staa, A. V. (2002). Uma Arquitetura Aberta para Geradores de Artefatos. *Simpósio Brasileiro de Engenharia de Software (SBES)*, pages 38–51.

Freitas, R. G. (2006). *Utilização de Geradores de Aplicação em Processos Ágeis de Reengenharia*. Dissertação de mestrado, Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP.

Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Geyer-Schulz, A. e Hahsler, M. (2002). Software reuse with analysis patterns. In *Proceedings of the 8th Americas Conference on Information Systems (AMCIS)*. Association for Information Systems.

Gomaa, H. e Shin, M. E. (2007). Automated software product line engineering and product derivation. In *Hawaii International Conference on System Sciences HICSS*. IEEE Computer Society.

Haley, E.R. Collins, G. C. D. (2007). The wonderful world of wiki benefits students and instructors. *Potentials, IEEE*, 27:21–26.

Harel, D. (2002). From Play-In Scenarios to Code: An Achievable. *Fundamental Approaches to Software Engineering (FASE)*, 1783:22–34.

Hass, A. M. J. (2003). *Configuration Management Principles and Practice*. Addison-Wesley Longman Publishing Co., Inc.

IEEE (1987). Std 1042 - IEEE Guide to Software Configuration Management.

IEEE (1990). Std 610.12 – IEEE Standard Glossary of Software Engineering Terminology.

IEEE (1996). IEEE/EIA Standard Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology Software Life Cycle Processes.

IEEE (2005). Std 828 – IEEE Standard for Software Configuration Management Plans.

ISO (1995). Quality Management – Guidelines for Configuration Management.

ISO (2004). ISO/IEC 15504 - Information technology - Process Assessment.

ISO e IEC (2003). International standard: ISO/IEC 12207.

Johnson, R. E. (1997). Frameworks = (components + patterns). *Communications of the ACM*, 40.

JUnit (2007). JUnit.org Resources for Test Driven Development. Disponível em: *http://www.junit.org/*. (Acessado 11 de Dezembro de 2007).

Keyes, J. (2004). *Software Configuration Management*. AUERBACH.

Krikhaar, R. L. e Crnkovic, I. (2007). Software configuration management. *Science Computer Program*, 65(3):215–221.

Kruchten, P. (2000). *The Rational Unified Process: An Introduction*. Addison-Wesley. 2° Edição.

Krueger, C. W. (2000). Software product line reuse in practice. *IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pages 117–118.

Leon, A. (2000). *A Guide to Software Configuration Management*. Artech House Publishers.

Linden, F. J., Schmid, K., e Rommes, E. (2007). *Software Product Lines in Action*. Springer.

Louridas, P. (2006). Using Wikis in software development. *IEEE Software*, 23(2):88–91.

Mason, M. (2006). *Pragmatic Version Control: Using Subversion (The Pragmatic Starter Kit Series)*. Pragmatic Bookshelf.

Mozilla Organization (2008). Bugzilla. Disponível em: *http://www.bugzilla.org/*. (Acessado 01 de Agosto de 2008).

Murta, L. G. P. (2006). *Gerência de Configuração no Desenvolvimento Baseado em Componentes*. Tese de doutorado, Universidade Federal do Rio de Janeiro, UFRJ, Brasil.

MySQL (2008a). MySQL: The world's most popular open source database. Disponível em: *http://www.mysql.com/*. (Acessado 13 de Junho de 2008).

MySQL (2008b). MySQL: The world's most popular open source database. Disponível em: *http://dev.mysql.com/downloads/connector/j/5.1.html*. (Acessado 13 de Junho de 2008).

Netbeans (2008). Netbeans IDE. Disponível em: *http://www.netbeans.org/*. (Acessado 30 de Julho de 2008).

Omondo (2008). Omondo: The Live UML Company. Disponível em: *http://www.eclipsedownload.com/*. (Acessado 26 de Abril de 2008).

Oracle (2008). Oracle Brasil. Disponível em: *http://www.oracle.com/global/br/index.html*. (Acessado 24 de Junho de 2008).

Pazin, A. (2004). *GAwCRe: um gerador de aplicações baseadas na web para o domínio de clínicas de reabilitação*. Dissertação de mestrado, Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP.

PMD (2008). PMD: Don't Shoot the Messenger. Disponível em: *http://pmd.sourceforge.net/*. (Acessado 25 de Abril de 2008).

Pohl, K., Böckle, G., e Linden, F. V. D. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Berlin Heidelberg New York.

Prange, H. F. (2007). *Uma avaliação empírica de um ambiente favorável para o desenvolvimento dirigido por testes*. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

Pressman, R. S. (2006). *Engenharia de Software*. McGraw-Hill.

Rizzo, J. A. (2005). *Validação de Processos Ágeis de Reengenharia e de Geradores de Aplicações*. Relatório de iniciação científica, Programa de Pós Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos – SP.

Rosenberg, L. H. (1996). Software re-engineering – technical report. Disponível em: *http://satc.gsfc.nasa.gov/support/reengrpt.PDF*. (Acessado 29 de Março de 2007).

Schäfer, W. (1996). Product-line development requires sophisticated software configuration management. *International Software Process Workshop (ISPW)*, page 15.

Schmidt, D. C., Fayad, M., e Johnson, R. E. (1996). Software patterns. *Communications of the ACM*, 39(10):37–39.

Sharp, A. (1997). *Smalltalk by Example*. McGraw-Hill.

Smaragdakis, Y. e Batory, D. (2000). Application Generators. Disponível em: *http://citeseer.ist.psu.edu/smaragdakis00application.html*. (Acessado 26 de Junho de 2008).

SOFTEX (2007). MPS.BR – Melhoria de Processo do Software Brasileiro - Guia Geral (Versão 1.1). Disponível em: *http://www.softex.br/mpsBr/_guias/MPS.BR_Guia_Geral_V1.2.pdf*.

Staples, M. (2004). Change Control for Product Line Software Engineering. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 572–573. IEEE Computer Society.

Sun Microsystems, Inc. (2008). J2EE – JavaServer Pages Technology. Disponível em: *http://java.sun.com/products/jsp/*. (Acessado 25 de Abril de 2008).

Thao, C., Munson, E. V., e Nguyen, T. N. (2008). Software configuration management for product derivation in software product families. In *International Conference and Workshop on the Engineering of Computer Based Systems ECBS*. IEEE Computer Society.

Trac (2008). Trac: Integrated SCM & Project Management. Disponível em: *http://trac.edgewall.org/*. (Acessado 4 de Maio de 2008).

W3C (2008). WC3: World Wide Web Consortium. Disponível em: *http://www.w3.org/*. (Acessado 01 de Agosto de 2008).

Weiss, D. M. e Lai, C. T. R. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley.

Yoder, J. W., Johnson, R. E., Wilson, Q. D., e Douglas, M. (1998). Connecting Business Objects to Relational Databases. *Fifth Conference on Patterns Languages of Programs (PLoP '98)*.

Yu, L. e Ramaswamy, S. (2006). A Configuration Management Model for Software Product Line. In *INFOCOMP – Journal of Computer Science*, volume 5, pages 1–8. Universidade Federal de Lavras.