PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks

Maria Istela Cagnin

PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks

Maria Istela Cagnin

Orientador: ***Prof. Dr. José Carlos Maldonado***

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, como parte dos requisitos para obtenção do título de Doutor em Ciências de Computação e Matemática Computacional.

"VERSÃO REVISADA APÓS A DEFESA"

USP - São Carlos
Setembro/2005

Aos meus pais, Genil e Maria José.

Agradecimentos

Agradeço a Deus pela vida e por ter me dado forças para concluir mais uma etapa de minha vida.

Ao Prof. José Carlos Maldonado, pelo apoio e dedicação durante a orientação desta tese e pela confiança em mim depositada.

Aos Profs. Fernão Stella Germano, Rosângela Penteado, Rosana Terezinha Vaccare Braga e Paulo Cesar Masiero, que sempre prontamente colaboraram neste trabalho. Meu muito obrigada!

Ao Jeferson pela paciência, apoio e compreensão durante esta etapa de minha vida.

Aos meus pais Genil e Maria José pelo apoio constante durante toda a minha vida. Sem vocês, tudo seria mais difícil!

Aos meus irmãos Gilson Carlos e Ana Lúcia, aos meus cunhados Gislaine e Marcos, e em especial, aos meus queridos e adoráveis sobrinhos Gustavo, Gabriel, Letícia e Hugo pelos inesquecíveis momentos de alegria.

A todos os amigos do Labes pelo companheirismo e pelos momentos de descontração. Fiz questão de não citar nomes para não correr o risco de esquecer alguém. Como diz o poeta: "Cada um que passa em nossa vida deixa um pouco de si e leva um pouco de nós".

Aos alunos de iniciação científica, Alessandra Chan e Celso Martinez pela amizade, dedicação e responsabilidade na condução dos estudos de caso.

A todos os funcionários do ICMC, em especial à Ana Paula, Laura, Beth, Tatiana e Enza pelo apoio acadêmico.

À FAPESP pelo apoio financeiro.

Sumário

Resumo **xiii**

Abstract **xv**

1 Introdução **1**
1.1 Contexto 1
1.2 Motivação 5
1.3 Objetivos 7
1.4 Organização da Tese 8

2 Revisão Bibliográfica **9**
2.1 Considerações Iniciais 9
2.2 Reengenharia de Software 10
2.2.1 Conceitos 10
2.2.2 Processos/Abordagens no Domínio de Sistemas de Informação 14
2.3 Padrões e Linguagens de Padrões 17
2.3.1 Conceitos 17
2.3.2 Linguagens de Padrões de Análise Relevantes no Domínio de Sistemas de Informação 19
2.3.2.1 Uma Linguagem de Padrões para Gestão de Recursos de Negócios - GRN 20
2.3.2.2 Outros Padrões e Linguagens de Padrões de Análise 22
2.3.3 Resumo das Linguagens de Padrões Estudadas 23
2.4 Frameworks 24
2.4.1 Conceitos 24
2.4.2 Frameworks de Aplicação Relevantes no Domínio de Sistemas de Informação 28
2.4.2.1 Framework GREN 28
2.4.2.2 Outros Frameworks 34
2.4.3 Resumo dos Frameworks Estudados 36
2.5 Métodos Ágeis 37
2.6 Processos de Software 44
2.6.1 Documentação de Processos de Software 45
2.7 Garantia da Qualidade de Software 51

2.7.1 Critérios de Teste Funcional: Particionamento de Equivalência e Análise do Valor Limite . . . 53
2.7.2 Teste de Software no Contexto de Métodos Ágeis . . . 53
2.8 Considerações Finais . . . 56

3 ARA: Um Arcabouço de Reengenharia Ágil 59
3.1 Considerações Iniciais . . . 59
3.2 ARA: Arcabouço de Reengenharia Ágil . . . 60
3.3 Processo PARFAIT . . . 61
3.4 Estudos de Caso para Refinar o PARFAIT . . . 78
3.5 Um Estudo de Caso para Avaliar o PARFAIT . . . 80
3.6 Necessidades Evidenciadas a partir dos Estudos de Caso . . . 86
3.7 Considerações Finais . . . 87

4 PREF: Um Processo de Evolução de Frameworks de Aplicação 89
4.1 Considerações Iniciais . . . 89
4.2 Processo PREF . . . 90
4.3 Uso do Processo PREF . . . 104
4.4 Considerações Finais . . . 112

5 ARTe: Uma Abordagem de Reúso de Teste baseada na Linguagem de Padrões de Análise GRN 115
5.1 Considerações Iniciais . . . 115
5.2 A Abordagem de Reúso de Teste ARTe . . . 116
5.2.1 Estratégia para Definir Recursos de Teste . . . 118
5.2.2 Diretrizes para Reutilizar Recursos de Teste . . . 124
5.3 Uso da Abordagem ARTe em uma Linguagem de Padrões de Análise . . . 126
5.3.1 Uso da Estratégia de Definição de Recursos de Teste . . . 127
5.3.2 Uso das Diretrizes para Reúso dos Recursos de Teste - Um Estudo de Caso . . . 135
5.4 Considerações Finais . . . 142

6 *GREN-WizardVersionControl*: Uma Ferramenta de Apoio ao Controle de Versões no PARFAIT 145
6.1 Considerações Iniciais . . . 145
6.2 Ferramenta *GREN-WizardVersionControl* . . . 146
6.2.1 Modelo Conceitual . . . 149
6.2.2 Arquitetura e Implementação . . . 150
6.3 Evolução da Ferramenta GREN-Wizard . . . 152
6.4 Um Exemplo de Uso da Ferramenta *GREN-WizardVersionControl* . . . 154
6.5 Considerações Finais . . . 161

7 Conclusão 163
7.1 Considerações Iniciais . . . 163
7.2 Resumo do Trabalho Realizado . . . 163
7.3 Contribuições . . . 164
7.4 Limitações do Trabalho Realizado . . . 167
7.5 Sugestões de Trabalhos Futuros . . . 168

Referências Bibliográficas **171**

A Pacote de Experimentação **191**
A.0.1 Fase de Definição 192
A.0.2 Fase de Planejamento 192
A.0.3 Fase de Operação 196

B Formulário para Levantamento de Dados do Engenheiro de Software **199**

C Formulário de Consentimento **205**

D Formulário de Coleta de Dados - Etapa 1 **207**

E Formulário de Coleta de Dados - Etapa 2 **215**

Lista de Figuras

2.1 Abordagens de reengenharia discutidas por Rosenberg (1996) 13
2.2 Estrutura da linguagem de padrões GRN (Braga et al., 1999) 21
2.3 Arquitetura do GREN (adaptada de (Braga, 2003)) 30
2.4 Hierarquia parcial de classes do GREN (Braga, 2003) 31
2.5 Arquitetura da ferramenta GREN-Wizard (Braga, 2003) 32
2.6 Tela da ferramenta GREN-Wizard 33
2.7 Níveis de modelagem (adaptado de (OMG, 2005)) 47
2.8 Pacote da estrutura do processo (OMG, 2005) 48
2.9 Estrutura do RUP (adaptado de Kruchten (2000)) 49
2.10 Ciclos de evolução (Kruchten, 2000) 49

3.1 Abordagem ARA 60
3.2 Visão Geral do PARFAIT 65

4.1 Visão Geral do PREF 91
4.2 Grafo do controle de versões do GREN e da ferramenta GREN-Wizard . . . 111
4.3 Grafo do controle de versões do GREN e dos sistemas gerados 112

5.1 Visão geral da abordagem ARTe (adaptada de Cagnin et al. (2004c)) 117
5.2 Padrão 4 da GRN – Locar o Recurso 127
5.3 Comparação dos dados de dois estudos de caso 141

6.1 Tela principal da ferramenta *GREN-WizardVersionControl* 148
6.2 Tela que apóia a inserção de novos atributos 149
6.3 Modelo conceitual da ferramenta *GREN-WizardVersionControl* 150
6.4 Arquitetura do framework GREN (adaptada de Braga (2003)) 151
6.5 Arquitetura das ferramentas GREN-Wizard e *GREN-WizardVersionControl* 152
6.6 Diagrama de classes da evolução da GREN-Wizard 153
6.7 Tela para solução de conflito 154
6.8 Diagrama de classes do sistema de biblioteca (Chan et al., 2003) 156
6.9 Tela da ferramenta GREN-Wizard durante a especificação do sistema de biblioteca 156
6.10 Inserção de novos atributos na classe `Livro` 157
6.11 Tela de empréstimo de livros e *script* da tabela `Emprestimo` 157
6.12 Sobreposição do método `EmprestimoSpec` na classe `EmprestimoForm` 158
6.13 Novo *layout* da tela de empréstimo de livros 158

6.14 Alteração do método `emprestimo` da classe `BibliotecaMainWindow` 159
6.15 Consiste remoção de elementos herdados do framework 159
6.16 Tela da ferramenta GREN-Wizard durante a especificação da multa 160
6.17 Tela de conflito da *GREN-WizardVersionControl* 161
6.18 Mensagem da ferramenta *GREN-WizardVersionControl* antes de gerar a base de dados do sistema 161
6.19 Tela de cadastro de multa e o *script* da respectiva tabela 162

7.1 Formas de reúso do PARFAIT 165

Lista de Tabelas

2.1 Resumo dos processos/abordagens de engenharia reversa e reengenharia . . 15
2.1 Resumo dos processos/abordagens de engenharia reversa e reengenharia (continuação) . . . 16
2.2 Resumo das linguagens de padrões de análise estudadas . . . 24
2.3 Resumo das classificações de frameworks . . . 27
2.4 Resumo dos frameworks estudados . . . 37
2.5 Fragmento da tabela de mapeamento de terminologias (OMG, 2005) . . . 51

3.1 Resumo da fase de concepção do PARFAIT . . . 68
3.2 Resumo da fase de elaboração do PARFAIT . . . 69
3.3 Resumo da fase de construção do PARFAIT . . . 70
3.4 Resumo da fase de transição do PARFAIT . . . 71
3.5 Atividades do PARFAIT para alcançar o objetivo das suas disciplinas . . . 71
3.6 Documentação de Classes de Equivalência (Myers, 2004a) . . . 73
3.7 Documentação dos casos de teste . . . 73
3.8 Coleta de dados do estudo de caso para avaliar o PARFAIT (Cagnin et al., 2003b) . . . 84
3.9 Outros dados coletados durante o estudo de caso . . . 85

4.1 Análise dos requisitos não cobertos pelo framework . . . 94
4.2 Gabarito do Histórico de Requisitos . . . 96
4.3 Histórico de Requisitos (Cagnin et al., 2004f) . . . 106
4.3 Histórico de Requisitos (Cagnin et al., 2004f) (continuação) . . . 107
4.4 Análise dos requisitos não cobertos pelo framework GREN (Cagnin et al., 2004d) . . . 108

5.1 Estratégia para associar recursos de teste a linguagens de padrões . . . 118
5.2 Estratégia para associar recursos de teste a LPAs . . . 119
5.3 Gabarito da documentação das classes de equivalência de consistência e de integridade . . . 121
5.4 Gabarito da documentação das classes de equivalência do negócio . . . 122
5.5 Diretrizes para reúso de recursos de teste (adaptadas de Cagnin et al. (2004c)) 125
5.6 Classes de equivalência de consistência e de integridade . . . 128
5.7 Documentação parcial dos requisitos de teste de consistência e de integridade dos padrões da GRN . . . 130

5.8 Classes de equivalência para o requisito de teste do negócio da classe `Locação de Recurso` do padrão 4 da GRN 131
5.9 Documentação parcial dos requisitos de teste do negócio da classe `Locação de Recurso` do Padrão 4 da GRN 132
5.10 Casos de teste de consistência e de integridade dos padrões da GRN 133
5.11 Casos de teste do negócio do padrão `Locação de Recurso` da GRN 134
5.12 Mapeamento dos casos de teste 135
5.13 Coleta de dados do estudo de caso para avaliar o reúso de teste 139
5.14 Outros dados coletados durante o estudo de caso 140

A.1 Resumo da fase de definição do pacote de experimentação 192
A.2 Projeto do experimento 194

Resumo

A necessidade de evolução de sistemas legados tem aumentado significativamente com o surgimento de novas tecnologias. Para apoiar essa tendência, diversos métodos de reengenharia têm sido propostos. No entanto, poucos possuem apoio computacional efetivo, alguns utilizam padrões de projeto ou padrões específicos de reengenharia, e nenhum utiliza framework baseado em linguagem de padrões. Este trabalho está inserido no domínio de Sistemas de Informação. Propõe a elaboração de um arcabouço de reengenharia ágil baseado em framework, que realiza a engenharia reversa do sistema legado com o apoio de linguagem de padrões de análise, fornecendo entendimento e documentação necessários para instanciar o framework. O entendimento do sistema legado também é apoiado pela sua execução, por meio de casos de teste. Esses casos de teste são utilizados posteriormente para validar o sistema alvo. O framework, cuja construção é baseada em linguagem de padrões, é utilizado para obter o projeto e a implementação do sistema alvo. Para permitir a reengenharia com o apoio do arcabouço definido, um processo ágil de reengenharia foi criado. Como no desenvolvimento de software, grande parte do tempo da reengenharia é despendido com atividades de VV&T. Para minimizar esse problema, uma abordagem de reúso de teste é proposta. Essa abordagem agrega recursos de teste aos padrões da linguagem de padrões de análise, permitindo o reúso, não somente das soluções de análise, como também dos recursos de testes associados. O uso de framework na reengenharia de software colabora para a sua evolução, pois o domínio ao qual pertence pode evoluir, já que nem todos os requisitos do domínio do framework podem ter sido elicitados durante o seu desenvolvimento. Assim, nesta tese é proposto também um processo de evolução de frameworks de aplicação. Os processos e a abordagem propostos são associados ao arcabouço definido para apoiar sua efetividade. Além disso, para avaliar o processo ágil de reengenharia, que fornece reúso em diversos níveis de abstração, um pacote de experimentação também é parcialmente definido. Estudos de caso e exemplos de uso foram conduzidos com os produtos definidos. Ressalta-se que outros estudos devem ser conduzidos para permitir a determinação de resultados com significância estatística.

Abstract

The need to evolve legacy systems has increased significantly with the advent of new technologies. To support this tendency, several reengineering methods have been proposed. However, few have effective computing support, some use design patterns or reengineering specific patterns and none use pattern language-based frameworks. This thesis's theme belongs to the Information Systems domain. An agile framework based reengineering infrastructure is proposed for the legacy system reverse engineering with the support of an analysis pattern language; also provided the understanding and documentation necessary for framework instantiation. The legacy system understanding is also supported by its execution with test cases. These are also subsequently used to validate the target system. The framework, whose construction is based on the analysis pattern language, is used to obtain the target system design and implementation. To allow the reengineering with the infrastructure support, an agile reengineering process has been created. As in software development, a large portion of the reengineering time is spent with VV&T activities. To minimize this problem, a testing reuse approach is proposed in this thesis. This approach aggregates test resources to the patterns of the analysis pattern language allowing reuse, not only of the analysis solutions, but also of the associated test resources. The framework used in software reengineering contributes to its evolution, as the domain to which they belong may evolve, and some of the framework domain requirements might not have been elicited during its development. Thus, in this thesis, a process for application framework evolution is also proposed. The processes and the approach are associated to the infrastructure defined to support its effectiveness. Furthermore, to evaluate the agile reengineering process that provides reuse at several abstraction levels, an experimentation package is also partially defined. Case studies and examples of use have been conducted with the products defined. We stress that other studies have to be done to enable the determination of results with statistical significance.

Capítulo

1

Introdução

1.1 Contexto

O software é um produto que evolui constantemente para satisfazer às necessidades de seus usuários. Para isso é necessário submetê-lo a constantes atividades de manutenção, que podem degradar o código fonte, tornando cada vez mais difícil mantê-lo e que, na maioria das vezes, não atualizam a documentação do software, culminando em uma situação em que a única documentação confiável é o próprio código fonte. Sistemas com essas características são denominados sistemas legados. É grande o interesse das empresas em manter esses sistemas em funcionamento, pois são considerados como propriedades e agregam lógicas do negócio codificadas, investimento, anos de desenvolvimento e teste, experiências, estratégias empresariais, etc.

As atividades de manutenção são geralmente realizadas quando há necessidade de: 1) adicionar funcionalidade que satisfaça a novas regras do negócio e/ou a novas políticas governamentais; 2) adaptar o sistema a novas tecnologias emergentes, tanto de hardware quanto de software; 3) corrigir erros que foram introduzidos durante o desenvolvimento ou manutenções anteriores do sistema; e 4) adicionar melhorias no sistema a fim de facilitar manutenções futuras e melhorar a sua confiabilidade.

Dentre as atividades de manutenção realizadas em um determinado software, a manutenção perfectiva representa um bom percentual, senão o maior. De acordo com Pressman (2005), a atividade de manutenção consome 70% de todo o esforço despendido durante o ciclo de vida do software. Muitas vezes, a baixa qualidade do software faz com que essa

atividade torne-se inviável de ser praticada. A grande parcela do esforço da atividade de manutenção é despendida no entendimento do software, que muitas vezes é dificultado pela não utilização de qualquer tipo de técnica de desenvolvimento durante a sua produção e a falta de qualquer tipo de documentação disponível ou atualizada.

Muitos autores contribuíram com métodos e técnicas para apoiar o desenvolvimento de software: métodos de análise estruturada clássica no final da década de 70 e início da de 80 (DeMarco, 1979; Gane e Sarson, 1982; Jackson, 1983; Yourdon e Constantine, 1978), análise essencial no final da década de 80 e início da de 90 (McMenamim e Palmer, 1991; Yourdon, 1989) e análise orientada a objetos na década de 90 (Booch, 1991; Coad e Yourdon, 1991; D. Coleman et al, 1994; Fowler e Scott, 1997; Jacobson, 1995). Infelizmente, nem sempre essas técnicas são utilizadas e, muitas vezes, são empregadas incorretamente, resultando em um produto com baixa qualidade, que não atende a todos os requisitos do usuário, com baixa manutenibilidade e difícil de ser entendido até mesmo por pessoas que participaram do seu desenvolvimento.

Na década de 80, com o surgimento da programação visual e, posteriormente, da programação orientada a objetos, que possibilitam o desenvolvimento de interfaces gráficas (*Graphical User Interface - GUI*) e o reúso de software, respectivamente, diversas empresas resolveram migrar seus sistemas legados para tais tecnologias. Essa atividade de migração está cada vez mais intensa devido à baixa usabilidade das interfaces textuais desses sistemas e pela disseminação das linguagens de programação visual, que utilizam o paradigma orientado a objetos.

Outro fator que contribui para a migração ou evolução dos sistemas legados está relacionado à forma de armazenamento dos dados. O armazenamento das informações dos sistemas legados é, em muitos casos, feito em sistemas de arquivos, sem nenhum tipo de controle de segurança, de integridade, de concorrência e de consistência. Isso causa riscos: financeiro, operacional e estratégico, pois toda a estória do negócio está armazenada de maneira não confiável. Uma forma de evitar esses riscos é a mudança da forma de armazenamento, por exemplo, de um sistema de arquivo para um Sistema de Gerenciamento de Banco de Dados (SGBD).

Outro tipo de mudança que está ocorrendo freqüentemente é a de plataforma. Muitas empresas estão migrando seus sistemas para plataformas de software livre (Perens, 1999), mais confiáveis e também com baixo ou nenhum custo de aquisição.

Diversas iniciativas de abordagens para apoiar a migração de sistemas legados para novas tecnologias (novas linguagens, SGBD's, novas plataformas e sistemas operacionais) têm surgido (Bisbal et al., 1999; Rosenberg, 1996; Tilley, 1995; Weiderman et al., 1997). Essa migração é denominada reengenharia de software, que consiste na análise e alteração do sistema, para reconstruí-lo em uma nova forma (Chikofsky e Cross, 1990) e, em geral, realiza uma combinação de outros processos: engenharia reversa (analisa um sistema existente, identifica seus componentes e os representa em um nível mais alto de abstração), reestruturação

ou refatoração (Fowler et al., 1999) (transformação de uma maneira de representação do sistema para outra no mesmo nível de abstração, preservando o seu comportamento externo, isto é, a sua funcionalidade e semântica), e engenharia avante (transferência das abstrações, modelos lógicos e projeto do sistema para uma implementação física).

Reengenharia é executada em oposição a construir um novo sistema devido aos procedimentos e lógicas do negócio do sistema que estão embutidos no código fonte do software, bem como ao não desperdício dos custos de desenvolvimento e manutenção (Rosenberg, 1996). Além disso, é possível que seja feita incrementalmente, até englobar todo o sistema, ou seja, o engenheiro de software pode realizar a reengenharia priorizando as partes do código que estão mais difíceis de ser mantidas ou que são mais importantes para o negócio e, em seguida, aplicá-la no restante do código. A maior vantagem da reengenharia é que ela preserva a solução existente do negócio em uma nova arquitetura técnica (Sneed, 1995) e é uma solução que deve ser considerada pelas empresas, pois os custos de desenvolvimento e implantação de um novo sistema podem ser muito altos (Warren e Ransom, 2002). Estudos têm mostrado que reengenharia, quando aplicada apropriadamente, geralmente promove custo efetivo e menos riscos do que o desenvolvimento de um novo sistema ((Ulrich, 1990) **apud** (Warren e Ransom, 2002)).

Esta tese está inserida no domínio de Sistemas de Informação (Laudon e Laudon, 2002), motivado pelo grande número de sistemas legados que pertencem a esse domínio e que necessitam ser submetidos à reengenharia. Além disso, há também o aspecto comercial relacionado ao interesse da indústria de software no grande número de sistemas desenvolvidos e em desenvolvimento que pertencem a tal domínio (MCT, 2002).

Do ponto de vista de reúso, padrões de software (Buschmann et al., 1996; Coad, 1992; Coplien, 1998; Gamma et al., 1995) e frameworks (Appleton, 1997; Buschmann et al., 1996; Fayad e Johnson, 2000; Markiewicz e Lucena, 2001; Schmidt e Buschmann, 2003; Taligent, 1997) estão sendo cada vez mais utilizados. Padrões de software colaboram para aumentar a produtividade das equipes. Além de fornecer soluções para problemas recorrentes, embutem conhecimento e experiência de especialistas. Frameworks permitem o reúso de grandes estruturas em um domínio particular e são personalizados para atender aos requisitos de aplicações específicas desse domínio (Braga, 2003; Johnson e Foote, 1988; Pree, 1995, 1999).

Observando todos esses aspectos e o reúso de software em diversos níveis de abstração, com o apoio de padrões de software e frameworks, proporcionando economia de tempo e esforço no desenvolvimento de software, evidenciou-se a possibilidade de utilizá-los nesta tese para apoiar a reengenharia.

Padrões específicos de engenharia reversa e reengenharia (Demeyer et al., 2000; Goedicke e Zdun, 2002; Lemos, 2002; Recchia, 2002; Stevens e Pooley, 1998), bem como padrões de projeto (Gamma et al., 1995) aplicados no contexto de reengenharia (Cha et al., 2004; Chu et al., 2001, 2000; Gall et al., 1996; Tahvildari e Kontogiannis, 2002; Tahvildari et al., 2003), ((Arsanjani, 2001a) **apud** (Cha et al., 2004)) são encontrados na literatura. Nesse último

caso, padrões de projeto são utilizados para apoiar o entendimento da estrutura e da arquitetura dos sistemas legados, tornando o sistema alvo (sistema resultante da reengenharia) mais flexível e reusável. Por outro lado, o entendimento da funcionalidade do sistema depende da experiência do engenheiro de software no domínio do sistema legado. Isso pode comprometer a reengenharia quando não houver esse tipo de profissional disponível.

A partir desse panorama e do conhecimento do domínio embutido em linguagens de padrões de análise (Arsanjani, 2001b; Braga et al., 1999; Pazin, 2004; Pazin et al., 2004; Ré, 2002; Ré et al., 2001; Ramesh, 1998), optou-se por utilizá-las nesta tese para apoiar o entendimento da funcionalidade do sistema legado. Frameworks, cuja construção tenha sido baseada em linguagens de padrões de análise, também foram selecionados para serem utilizados, a fim de apoiar tanto as atividades de engenharia reversa quanto as atividades de engenharia avante. Nesse tipo de framework, as classes e os relacionamentos contidos em cada padrão da linguagem de padrões de análise, possuem implementação correspondente na sua hierarquia de classes. As variantes de cada padrão são, em geral, implementadas como *hot spots* [1] no framework.

Outra carência notada é a necessidade de ferramentas de apoio computacional e atividades de VV&T (Validação, Verificação e Teste) associadas a processos e abordagens de reengenharia, sendo que a migração de sistemas antigos requer um alto grau de automação devido às dimensões completas e complexas da tarefa (Sneed, 1998) e a garantia de qualidade deve ser considerada para entregar um produto confiável a seus usuários (Maldonado e Fabbri, 2001a). Nesse contexto, outra carência observada refere-se à ausência de iniciativas de testes associados a padrões de software, nos diferentes níveis de abstração (por exemplo, análise, projeto e implementação), a fim de fornecer recursos de teste capturados a partir das soluções embutidas nos padrões. Isso pode colaborar para a redução do tempo gasto com atividades de teste tanto no desenvolvimento quanto na reengenharia baseados em padrões de software.

Além disso, como no desenvolvimento de software, na reengenharia há também a preocupação de entregar o software sem atraso em relação ao que foi previsto e com custo não superior ao que foi estimado. Nesse contexto, surgiram, na década de 90, os métodos ágeis (Ambler e Jeffries, 2002; Beck, 2000; Coad, 1999; Hunt e Thomas, 1999; K.Schwaber e Beedle, 2002; Stapleton, 1997). No entanto, nenhum processo de reengenharia que utilize as práticas de métodos ágeis foi encontrado na literatura até o momento da escrita desta tese.

É significativa a importância da engenharia de software experimental para validar estatisticamente as técnicas desenvolvidas e também a sua aplicabilidade em diferentes contextos e ambientes. Nesse interim, nenhum trabalho relevante de experimentação na área de reengenharia foi encontrado na literatura especializada. Porém, diversos trabalhos podem ser encontrados em outras áreas da Engenharia de Software (Braga et al., 2003; M. e Moreno, 2003; Shull et al., 2000), sendo que alguns são relacionados a frameworks (Braga et al., 2003; Shull et al., 2000).

[1] Maiores detalhes sobre *hot spots* são encontrados na Subseção 2.4.1 do Capítulo 2 desta tese.

O trabalho aqui proposto encaixa-se nesse contexto, definindo um arcabouço de reengenharia ágil com base em frameworks, cuja construção seja baseada em linguagem de padrões, com atividades de VV&T associadas, que facilite a evolução de um sistema legado procedimental para o paradigma orientado a objetos. Para permitir a aplicabilidade do arcabouço, diversos recursos foram criados e foram a ele associados, destacando um processo ágil de reengenharia. O trabalho preocupa-se também com a definição de um pacote de experimentação para apoiar a análise e evolução desse processo de reengenharia.

1.2 Motivação

Diversos métodos de engenharia reversa e reengenharia de sistemas procedimentais para o paradigma orientado a objetos existem na literatura, como a abordagem de Sneed e Nyáry (1995), o método COREM (Gall e Klösch, 1993), o método Fusion/RE (Penteado, 1996), o método Fusion-RE/I (Costa, 1997), o método de van Deursen e Kuipers (1998), o método Renaissance (Esprit Project 29512, 1999; Warren e Ransom, 2002), o método de Cimitile et al. (1999), o método PRE/OO (Lemos, 2002), o método IRLS (Bianchi et al., 2003), o processo de reengenharia de software baseada em padrões (Chu et al., 2001, 2000) e o processo de reengenharia baseado em arquitetura (Cha et al., 2004). Dentre esses, apenas alguns possuem apoio computacional parcial (Bianchi et al., 2003; van Deursen e Kuipers, 1998; Gall e Klösch, 1993; Sneed e Nyáry, 1995), poucos utilizam padrões de reengenharia (Lemos, 2002; Recchia, 2002) ou padrões de projeto no contexto de reengenharia (Cha et al., 2004; Chu et al., 2001, 2000), poucos consideram teste na reengenharia (Bianchi et al., 2003; Esprit Project 29512, 1999; Warren e Ransom, 2002) e nenhum utiliza linguagens de padrões de análise e frameworks de software.

Muitos vendedores de ferramentas de software divulgam ferramentas de conversão de linguagem (ou seja, convertem automaticamente um programa, escrito em uma linguagem, para outra) como uma solução para a migração dos sistemas legados procedimentais para orientados a objetos. De acordo com Terekhov e Verhof (2000), conversão de linguagem parece ser um negócio de risco, sendo que os problemas com os conversores vão desde a conversão sintática até à semântica (por exemplo, conversão de tipos, considerando que muitas vezes não existe a equivalência de tipos entre duas linguagens diferentes). A partir desses problemas, Terekhov e Verhof (2000) questionam se os programas convertidos são realmente mais fáceis de manter do que os programas originais, que foram submetidos à conversão.

A autora realizou, em seu programa de mestrado (processo FAPESP #97/12208-0) (Cagnin, 1999), a reengenharia com e sem o uso de padrões de software de um ambiente para edição e simulação de statecharts (Harel, 1987), denominado StatSim (Masiero et al., 1991). Esse ambiente foi originalmente desenvolvido em linguagem C e Xview, com

30.000 linhas de código e armazenamento das informações em arquivos de texto. Em uma primeira etapa do trabalho, realizou-se a reengenharia total do StatSim, mantendo a linguagem de programação original, no entanto, com a adição de características do paradigma orientado a objetos como: 1) migração dos módulos para "pseudo-classes", 2) transformação de procedimentos extensos em diversos "pseudo-métodos" e 3) encapsulamento de dados. Em uma segunda etapa, realizou-se a reengenharia parcial do StatSim, com uso de padrões de projeto de persistência (Yoder et al., 1998), da linguagem Java e do banco de dados relacional Sybase. A reengenharia, tanto da primeira quanto da segunda etapa, foi conduzida sem apoio computacional. Com isso, observou-se a necessidade e a importância de ferramentas que apóiem a reengenharia de sistemas.

É importante ressaltar que produzir software com alta qualidade e com baixo custo é uma das preocupações da Engenharia de Software. Desde a época da programação estruturada (década de 70) já se utilizavam módulos de código prontos para aumentar a produtividade. Com o surgimento da orientação a objetos, o reúso de código mostrou-se mais poderoso. Com o intenso crescimento da indústria de software, houve a necessidade de reúso em um nível maior de granularidade. A partir desse cenário, frameworks foram criados para permitir o reúso de estruturas maiores em um determinado domínio. Assim, famílias de aplicações similares, mas não idênticas, podem ser derivadas a partir de um único framework e o engenheiro de software deve se preocupar apenas com o desenvolvimento das partes do negócio que são específicas a cada empresa.

A utilização de frameworks baseados em linguagem de padrões para apoiar a reengenharia de sistemas legados é uma linha de estudo em aberto, que pode facilitar a migração de sistemas legados e pode evitar os problemas criados quando da utilização de conversores de linguagem e quanto à carência de ferramentas voltadas para tal migração. Além disso, o produto obtido após o processo de reengenharia, utilizando framework, possui boa qualidade, considerando que o framework tenha sido validado durante sua construção. Essa característica do produto é difícil de ser alcançada quando se utilizam conversores de linguagem, como declarado por Terekhov e Verhof (2000).

Um framework, denominado GREN (Braga, 2003), baseado em uma linguagem de padrões do domínio de Gestão de Recursos de Negócios, denominada GRN (Braga et al., 1999), foi desenvolvido em um trabalho de doutorado no Grupo de Engenharia de Software do Instituto de Ciências Matemáticas e de Computação (ICMC-USP). De acordo com Braga et al. (1999), o domínio de Gestão de Recursos de Negócios é um domínio particular de Sistemas de Informação e envolve o gerenciamento de recursos [2]. Muitos sistemas legados ainda são largamente utilizados no Brasil, mesmo tendo sido desenvolvidos com tecnologia considerada ultrapassada nos dias de hoje. A existência de frameworks no domínio de Gestão de Recursos de Negócios pode motivar e agilizar a reengenharia de tais sistemas (Braga, 2003). A partir dessa motivação e com a carência de ferramentas que apóiam efetivamente

[2]Recurso, nesse caso, é todo bem ou serviço que pode ser locado, mantido ou comercializado.

métodos e abordagens de reengenharia de sistemas, bem como com o uso de frameworks para apoiar o desenvolvimento de sistemas em novas tecnologias, é explorada nesta tese a utilização de frameworks, baseados em linguagem de padrões, mais especificamente o framework GREN, para apoiar a reengenharia de sistemas. Frameworks desse tipo favorecem reúso de análise, além de projeto e implementação.

O framework GREN é utilizado neste trabalho devido a diversas razões. É um framework baseado em linguagem de padrões de análise, que é de interesse desta tese, e pertence ao domínio de Gestão de Recursos de Negócios, que é familiar ao Grupo de Pesquisa e também à autora, pois durante a sua graduação e especialização em análise de sistemas concentrou-se nesse tipo de domínio. Além disso, esta tese foi desenvolvida no Grupo de Pesquisa do qual a autora do GREN faz parte. Isso facilita a troca de informação sobre o framework (desenvolvimento e utilização) e também sua evolução, necessária para que a reengenharia baseada em framework possa ser efetivamente realizada. Na implementação do framework GREN, padrões de projeto também foram utilizados (por exemplo, *Strategy*, *Factory Method*, *Template Method* (Gamma et al., 1995) e *PersistenceLayer* (Yoder et al., 1998)), o que colaborará para maior flexibilidade e manutenibilidade dos sistemas resultantes da reengenharia.

De acordo com os resultados de uma pesquisa sobre frameworks de aplicação realizada por Yassin e Fayad (2000), a porcentagem das classes dos frameworks utilizada na geração dos sistemas pode chegar a 90%. No entanto, a porcentagem média de uso dessas classes no sistema gerado é de aproximadamente 55%. Porém, para esses autores, essa não é a porcentagem média e sim a porcentagem mínima, pois eles acreditam que para frameworks que atendem amplamente o domínio ao qual pertencem, 75% das suas classes, no mínimo, são utilizadas nos sistemas gerados. Dessa maneira, quanto mais o sistema legado estiver inserido no domínio da linguagem de padrões e, conseqüentemente, do framework, maior será a porcentagem das classes do framework utilizada no sistema alvo, resultante da reengenharia. Com isso, reduzem-se as modificações manuais no sistema alvo obtido pela instanciação do framework, tornando o custo da reengenharia mais baixo e viável.

1.3 Objetivos

Esta tese tem como principal objetivo definir um arcabouço de reengenharia ágil baseado em framework, com reúso de teste, no domínio de Sistemas de Informação. Atividades de VV&T devem ser utilizadas desde o entendimento do sistema legado até a validação do sistema alvo. Um dos objetivos é concentrar-se no entendimento do sistema legado com o apoio da linguagem de padrões, utilizada na construção do framework, visando elaborar documentação orientada a objetos do sistema para instanciar o framework, produzindo uma versão do sistema alvo o mais rápido possível. Requisitos e regras do negócio do sistema

legado, não fornecidos pelo framework, devem ser implementados manualmente no código do sistema alvo para torná-lo com funcionalidade semelhante à do sistema legado. Para apoiar a efetividade do arcabouço, diversos recursos serão definidos e associados a ele, como, processos, ferramentas e abordagens de reúso. Outros recursos disponíveis serão também utilizados no contexto de reengenharia, como, práticas de métodos ágeis e frameworks baseados em linguagens de padrões de análise.

1.4 Organização da Tese

Este trabalho está organizado em sete capítulos e cinco apêndices, como descrito a seguir.

No Capítulo 2 apresenta-se a revisão bibliográfica realizada para o desenvolvimento e entendimento desta tese.

No Capítulo 3 apresenta-se a infraestrutura do arcabouço de reengenharia ágil definido, destacando o processo ágil de reengenharia baseado em framework criado e alguns estudos de caso conduzidos, que proporcionaram o refinamento e a evolução desse processo. Nos próximos três capítulos apresentam-se os recursos criados e associados ao arcabouço.

No Capítulo 4 apresenta-se a descrição do processo de evolução de frameworks de aplicação e exemplos de evoluções conduzidas no framework GREN com o apoio de tal processo.

No Capítulo 5 apresentam-se a abordagem de reúso de teste definida para linguagens de padrões de análise e um exemplo de uso dessa abordagem na linguagem de padrões GRN. Nesse capítulo apresenta-se também um estudo de caso de reengenharia com reúso de recursos de teste, proporcionado pela abordagem definida.

No Capítulo 6 apresenta-se uma ferramenta criada para apoiar o controle de versões no processo de reengenharia definido no Capítulo 3, objetivando contribuir com a sua iteratividade. Além disso, um exemplo de uso da ferramenta criada também é fornecido.

No Capítulo 7 apresentam-se o resumo do trabalho realizado, as suas limitações e contribuições. Sugestões de pesquisas futuras, decorrentes desta tese, também são discutidas nesse capítulo.

No Apêndice A apresentam-se a definição parcial de um pacote de experimentação no contexto de reengenharia e um estudo de caso conduzido para validar tal pacote. Finalmente, nos Apêndices B, C, D e E apresentam-se os formulários necessários para o uso do pacote definido.

Capítulo

2

Revisão Bibliográfica

2.1 Considerações Iniciais

Neste capítulo é apresentado o embasamento teórico necessário para a compreensão do trabalho desenvolvido nesta tese. Na Seção 2.2 são apresentados conceitos de reengenharia e alguns processos e abordagens de reengenharia encontrados na literatura, especificamente que apóiam a migração de sistemas procedimentais para o paradigma orientado a objetos, que é o principal foco desta tese. Nas seções seguintes, apresentam-se os outros temas envolvidos. Na Seção 2.3 são definidos os conceitos de padrão e de linguagem de padrões e são apresentadas algumas linguagens de padrões de análise pertinentes ao domínio de interesse desta tese. Na Seção 2.4 são definidos os conceitos de frameworks sob diversos aspectos e também são apresentados alguns frameworks de aplicação, tanto acadêmicos quanto comerciais, relacionados ao domínio tratado nesta tese. Na Seção 2.5 são apresentados os fundamentos de métodos ágeis, destacando as práticas do método ágil e**X**treme **P**rogramming. Essas práticas foram comparadas com as características do processo de reengenharia definido nesta tese, objetivando analisar sua agilidade [1]. Na Seção 2.6 é salientada a importância de processos de software, inclusive no contexto de reengenharia, e são discutidos alguns formatos de documentação de processos de software encontrados na literatura. Na Seção 2.7 são apresentados conceitos de teste de software, enfocando principalmente técnicas aplicadas no contexto de métodos ágeis. Finalmente, na Seção 2.8 são apresentadas as conclusões finais deste capítulo.

[1] Capacidade de se enquadrar como método ágil.

2.2 Reengenharia de Software

2.2.1 Conceitos

Diversas definições de reengenharia de software são encontradas na literatura (Jacobson, 1991; Pressman, 2005; Sommerville, 2000; Tilley, 1995). No entanto, o ponto de referência de todas elas é a definição de Chikofsky e Cross (1990). Os seis termos relacionados a reengenharia de software propostos por Chikofsky e Cross (1990) são apresentados a seguir:

- **engenharia avante:** transferência das abstrações, modelos lógicos e projeto do sistema para uma implementação física;
- **engenharia reversa:** processo de análise de um sistema legado, identifica seus componentes e os seus relacionamentos e os representa em um nível mais alto de abstração. Pode ser classificada como:
  - **redocumentação:** consiste na criação ou revisão de uma representação da abstração semântica do sistema (por exemplo, fluxo de dados e de controle, diagrama de entidades e relacionamentos e listagem de referência cruzada dos elementos do código); e
  - **recuperação do projeto:** considera domínio do conhecimento e informações externas para identificar no sistema abstrações de alto nível, além daquelas obtidas diretamente pela análise do sistema, a fim de fornecer completo entendimento sobre "o quê" o sistema faz, "como" ele faz e "por que" ele faz de uma determinada maneira;
- **reestruturação:** transformação de uma maneira de representação do sistema para outra no mesmo nível de abstração, preservando o seu comportamento externo, isto é, a sua funcionalidade e semântica. Esse termo também é conhecido como refatoração (Fowler et al., 1999);
- **reengenharia:** conhecida também como renovação, consiste na análise e alteração do sistema, para reconstruí-lo em uma nova forma. Geralmente inclui engenharia reversa seguida por engenharia avante ou reestruturação.

O objetivo de Chikofsky e Cross (1990) não foi a criação de novos termos mas a racionalização e a padronização dos termos que já estavam sendo usados. O termo "engenharia reversa" teve origem na análise de hardware por ser comum inferir projetos a partir de produtos finalizados. Embora o objetivo de engenharia reversa na área de hardware seja em geral duplicar o sistema, na área de software considera-se como objetivo a obtenção

do entendimento suficiente do sistema e de sua estrutura para apoiar a sua manutenção, a sua melhoria ou a sua substituição.

De acordo com Sneed (1995), os quatro objetivos principais de reengenharia são: 1) melhorar a manutenibilidade do sistema (por exemplo, realizar reengenharia em módulos menores), 2) migrar o sistema (por exemplo, movê-lo para um ambiente operacional mais barato ou melhor), 3) alcançar maior confiabilidade (por exemplo, reestruturar o sistema) e 4) preparar para aumentar a funcionalidade (por exemplo, decompor programas em módulos menores para tornar mais fácil modificar ou adicionar requisitos). A principal preocupação desta tese está relacionada à migração de sistemas legados procedimentais para o paradigma orientado a objetos no domínio de Sistemas de Informação.

Para Sneed (1995), a maior vantagem de reengenharia é que ela preserva a solução existente do negócio em uma nova arquitetura técnica. Warren e Ransom (2002) ressaltam que reengenharia é uma solução que deve ser considerada pelas empresas, pois os custos de desenvolvimento e implantação de um novo sistema pode ser muito alto, inviabilizando-o. Além disso, estudos têm mostrado que reengenharia, quando aplicada apropriadamente, geralmente promove custo efetivo e menos riscos do que desenvolver um novo sistema ((Ulrich, 1990) **apud** Warren e Ransom (2002)).

Rosenberg (1996) apresenta diversos riscos associados à reengenharia de software, que devem ser evitados, de alguma forma, pelos processos de reengenharia: não cumprimento do prazo e custo inicialmente estimados; seleção inadequada de subsistemas submetidos ao processo de reengenharia; gerenciamento de configuração inadequado; alto custo da reengenharia; ausência de garantia de qualidade do sistema resultante da reengenharia, o que causa a insatisfação dos usuários; grande extensão e alto custo da documentação produzida; dificuldade de recuperar o projeto e os requisitos a partir do código fonte; elicitação de objetos incompleta e incorreta; perda do conhecimento do negócio embutido no código fonte; recuperação de informação sem utilidade ou não utilizada; dificuldade na migração dos dados existentes; insuficiência da engenharia reversa; e ausência de conhecimento e experiência do pessoal envolvido na reengenharia.

Sistema legado é outro termo utilizado no contexto de reengenharia de software. Sistemas desse tipo ainda estão em funcionamento, mas sua qualidade e vida operacional é constantemente deteriorada devido a ocorrência de diversas atividades de manutenção e atualizações tecnológicas (Zou e Kontogiannis, 2002) e, na maioria das vezes, rodam em tecnologias de hardware e software ultrapassadas (Rahgozar e Oroumchian, 2003). Esse tipo de sistema possui geralmente documentação incompleta, incorreta ou até inexistente, tornando o código fonte a única documentação existente (Postema e Schmidt, 1998). Isto torna difícil entender o que o sistema está fazendo, como o trabalho é realizado e porque ele está codificado de uma determinada maneira. Consequentemente, sistemas legados são difíceis de serem modificados e as modificações são difíceis de serem validadas (Baxter e Mehlich, 1997). Por isso, estão sujeitos a serem submetidos a reengenharia.

Alguns dos principais fatores que desencadeiam necessidades de modificações nos sistemas legados são: mudanças nos requisitos dos usuários, aparecimento de novas tecnologias, mudanças nos objetivos do negócio e a necessidade das empresas explorarem novas oportunidades de mercado (Warren e Ransom, 2002).

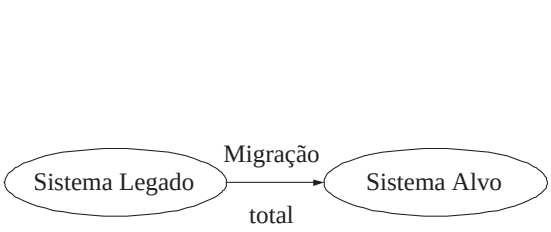
Existem diversas maneiras de conduzir a reengenharia de software ((Bisbal et al., 1999; Rosenberg, 1996; Tilley, 1995; Weiderman et al., 1997)), uma delas ((Rosenberg, 1996)) depende de quão crítico é o problema que deve ser resolvido. Nessa perspectiva, quatro abordagens foram propostas por Rosenberg (1996), sendo que têm como objetivo a preocupação em como conduzir a reengenharia: abordagem de reengenharia "big bang", incremental, evolutiva e híbrida.

A *Abordagem de Reengenharia "Big Bang"* troca o sistema legado pelo sistema alvo de uma só vez, como ilustrado na Figura 2.1(a), e é utilizada para solucionar problemas críticos, que devem ser resolvidos rapidamente. A vantagem é que nenhuma interface precisa ser criada entre os componentes velhos e os novos. As desvantagens são: 1) não é adequada para sistemas grandes e 2) as mudanças que ocorrem no sistema legado, cujas solicitações foram feitas durante a reengenharia, devem ser realizadas também no sistema alvo. Essa abordagem possui alto risco.

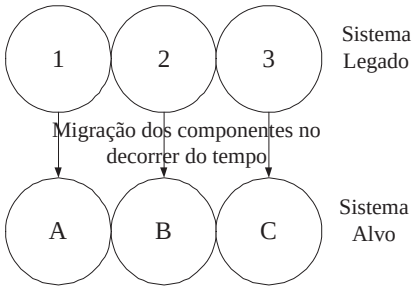
A *Abordagem de Reengenharia Incremental* troca o sistema legado pelo sistema alvo de maneira incremental, ou seja, os componentes do sistema legado são migrados, no decorrer do tempo, em versões do sistema alvo, como mostrado na Figura 2.1(b). As vantagens são: 1) como a reengenharia é feita em partes, ela torna-se mais rápida e é mais fácil encontrar erros e 2) as versões do sistema alvo são entregues aos usuários, à medida que são liberadas para uso, e esses realizam teste de aceitação. As desvantagens são: 1) necessidade de controle sistemático de configuração e 2) a estrutura geral do sistema alvo deve permanecer a mesma que a do sistema legado e somente a estrutura interna dos componentes do sistema alvo pode ser alterada. Essa abordagem tem risco menor do que a abordagem *"Big Bang"* porque é possível identificar e monitorar os riscos apenas no componente que está sendo submetido à reengenharia.

Como na abordagem *Incremental*, a *Abordagem de Reengenharia Evolutiva* baseia-se na migração de componentes do sistema legado para componentes do sistema alvo, como apresentado na Figura 2.1(c). A diferença é que a seleção dos componentes existentes para a migração é baseada nas funções em comum e não nas estruturas. A vantagem é que as funções semelhantes são concentradas em um único componente. As desvantagens são: 1) identificação de objetos com funções similares espalhados pelo código fonte do sistema legado, que devem ser agrupados em um único componente e 2) muito esforço para identificar objetos funcionais.

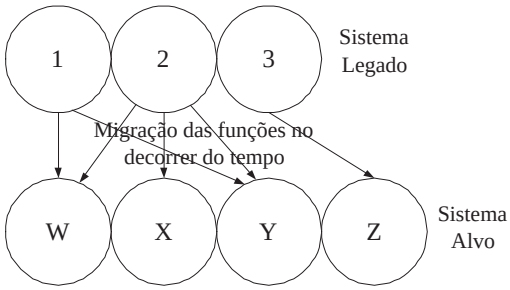
Abordagem de Reengenharia Híbrida migra o sistema legado para o sistema alvo com o apoio de níveis de abstração e métodos e técnicas específicas, como ilustrado na Figura 2.1(d). O código fonte do sistema legado é usado para: 1) recriar o projeto do sistema alvo; 2)



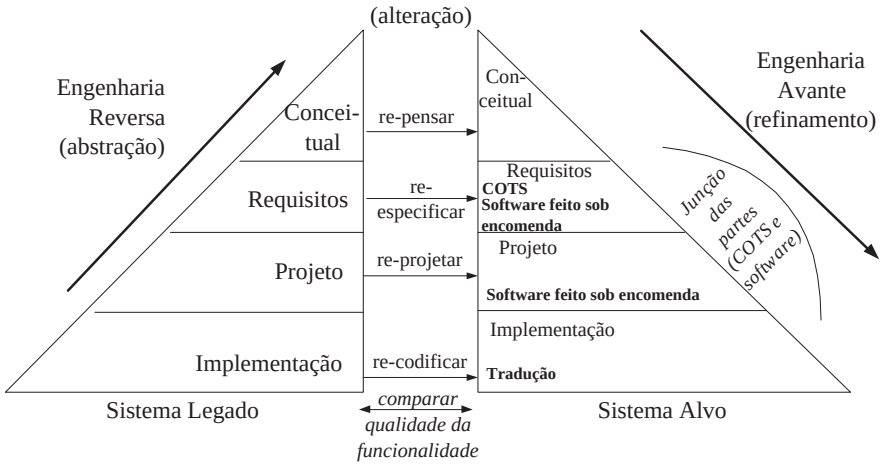
(a) Abordagem de Reengenharia "Big Bang"



(b) Abordagem de Reengenharia Incremental



(c) Abordagem de Reengenharia Evolutiva



(d) Abordagem de Reengenharia Híbrida

Figura 2.1: Abordagens de reengenharia discutidas por Rosenberg (1996)

identificar requisitos do sistema legado ou novos, e remover os que não estão mais sendo usados para re-especificá-los; 3) re-estruturar e re-projetar o sistema alvo (usando abordagem OO - Orientada a Objetos) e 4) codificar o sistema alvo. Essa abordagem utiliza três passos para realizar a reengenharia:

- **tradução**: traduz o sistema legado para uma nova linguagem de programação, sistema operacional ou plataforma de hardware;
- **COTS (*Commercial-Off-The-Shelf*)**: identifica requisitos do sistema existente que podem ser satisfeitos pelo reúso de pacotes COTS. É necessário identificar também os requisitos que são atendidos parcialmente ou que não são atendidos pelo COTS para determinar o quanto que o COTS deve ser adaptado;

- **software feito sob encomenda**: implementa os requisitos que não podem ser satisfeitos pelos outros dois passos para unir ("glue") as partes do sistema alvo, resultantes desses passos.

Após o término da reengenharia híbrida, é necessário verificar se a transição do sistema legado para o sistema alvo foi bem sucedida. Para isso, é sugerido executar os casos de teste do sistema legado no sistema alvo. As vantagens são: 1) o uso de pacotes COTS diminui o tempo de desenvolvimento e de teste e, então, os custos; e 2) o uso de COTS apropriados aumenta a confiabilidade do sistema alvo, uma vez que já foram testados anteriormente. As desvantagens são: 1) risco de interoperabilidade e de comunicação entre as interfaces dos componentes do sistema alvo resultantes dos diversos passos; e 2) seleção de critérios de teste adequados para realizar o teste de integração do sistema alvo.

2.2.2 Processos/Abordagens no Domínio de Sistemas de Informação

Diversos autores têm se preocupado com questões de engenharia reversa e reengenharia na área de Engenharia de Software, como pode ser observado na Tabela 2.1. O surgimento de padrões de software fez com que autores de diversos métodos e abordagens de reengenharia definissem padrões específicos, como é o caso de Recchia (2002) e Lemos (2002). Outros optaram por utilizar padrões de projeto existentes (Gamma et al., 1995), como é o caso de Gall e Klösch (1993), Chu et al. (2001, 2000) e Cha et al. (2004). Além disso, atividades de teste são consideradas explicitamente em poucos processos de reengenharia. Teste de sistema e teste de aceitação são considerados pelo Renaissance (Esprit Project 29512, 1999; Warren e Ransom, 2002). Teste de equivalência é utilizado em IRLS (Bianchi et al., 2003) para garantir que o comportamento do sistema alvo permanece o mesmo após a reengenharia de um ou mais componentes do sistema legado. A criação desse tipo de teste é baseada em exemplos de uso, considerados pelos usuários. Ressalta-se que a ausência de atividades de teste na reengenharia pode comprometer a qualidade do sistema resultante, podendo levar a reengenharia ao fracasso.

Com relação ao uso de padrões de projeto na reengenharia de sistemas, Gall et al. (1996) afirmam que a recuperação desse tipo de padrões em sistemas legados procedimentais é uma tarefa difícil, pois não se pode esperar encontrar uma instanciação direta de um padrão particular no código fonte, e essa tarefa torna-se um pouco mais complexa se o sistema foi desenvolvido por diversos programadores. Chu et al. (2000) afirmam que a tarefa mais difícil e dispendiosa é a migração do código legado para uma nova linguagem alvo, particularmente as estruturas de dados antigas em padrões de projeto. A partir disso, e com a necessidade de apoio computacional associado a processos e abordagens de reengenharia, como pode ser observado na Tabela 2.1, notou-se a importância do uso de frameworks na reengenharia de software. Isso porque não há necessidade de mapeamento de padrões de projeto no código fonte do legado e não há perigo de selecionar padrões de projeto incorretamente na

reengenharia pois, em geral, como padrões de projeto são utilizados na implementação de frameworks, eles são obtidos automaticamente a partir da sua instanciação; e a migração do código legado para uma nova linguagem alvo é praticamente automática com o uso de frameworks, pois somente os requisitos e regras do negócio específicos do sistema legado são implementados manualmente no sistema alvo.

Tabela 2.1: Resumo dos processos/abordagens de engenharia reversa e reengenharia

Processo/ Abordagem	**Objetivo**	**Apoiado por Computador?**	**Escopo**
COREM (Gall e Klösch, 1993, 1995)	Realiza a reengenharia orientada a objetos de sistemas legados procedimentais, utilizando conhecimento do domínio da aplicação e padrões de projeto para apoiar a engenharia reversa.	Parcial (ferramenta CORET (Taschwer et al., 1999))	Reengenharia
Não possui nome específico (Sneed e Nyáry, 1995)	Extrai "automaticamente" a documentação de projeto orientada a objetos de programas em COBOL e realiza a reengenharia manual.	Parcial (ferramenta OBJECT-REORG)	Reengenharia
Fusion/RE (Penteado, 1996)	Utiliza os modelos de análise do método Fusion (D. Coleman et al, 1994) para obter uma pseudo documentação OO do sistema legado, aprimorando-a depois para torná-la efetivamente OO.	Não	Engenharia Reversa
Fusion-RE/I (Costa, 1997)	Baseado no método Fusion/RE. Elabora a documentação orientada a objetos de sistemas legados procedimentais a partir de aspectos operacionais e de dados disponíveis na interface com o usuário.	Não	Engenharia Reversa
Não possui nome específico (van Deursen e Kuipers, 1998)	Identifica objetos (atributos e métodos) das estruturas de dados legados de programas em COBOL por meio da reestruturação semi-automática dessas estruturas.	Parcial	Engenharia Reversa
Renaissance (Esprit Project 29512, 1999; Warren e Ransom, 2002)	Fornece diversas estratégias de evolução do sistema legado, que são selecionadas observando os fatores de risco de cada estratégia. A evolução é guiada por um modelo de processo que contém tanto o planejamento da evolução quanto o gerenciamento do seu projeto.	Não	Reengenharia
Não possui nome específico (Cimitile et al., 1999)	Decompõe sistemas legados em objetos. Os métodos são associados aos objetos por intermédio de métricas de projeto orientadas a objetos.	Não	Engenharia Reversa

Tabela 2.1: Resumo dos processos/abordagens de engenharia reversa e reengenharia (continuação)

Processo/ Abordagem	**Objetivo**	**Apoiado por Computador?**	**Escopo**
Reengenharia de software baseada em padrões (Chu et al., 2001, 2000)	Migra sistemas legados desenvolvidos com abordagens de projeto tradicionais para sistemas baseados em padrões de projeto visando melhorar a sua capacidade de entendimento, reúso e manutenção.	Não	Reengenharia
FaPRE/OO (Recchia, 2002)	Evolução do Fusion/RE, é baseado na abordagem incremental e em padrões de reengenharia.	Não	Reengenharia
PRE/OO (Lemos, 2002)	Evolução da FaPRE/OO com a criação de dois *clusters* específicos de padrões para tratar a qualidade do processo e do produto.	Não	Reengenharia
IRLS (*Iterative Reengineering of Legacy Systems*) (Bianchi et al., 2003)	IRLS realiza reengenharia gradual em sistemas legados procedimentais, a partir do re-projeto da base de dados.	Parcial (Microfocus Revolve tool 5.0 (MERANT, 2000))	Reengenharia
Processo de Reengenharia baseado em Arquitetura (Cha et al., 2004)	Define uma arquitetura alvo, refinando informação da arquitetura do sistema legado extraída por meio de análise do domínio; identifica os padrões de reengenharia (baseados nos de Gamma et al. (1995)) que são aplicáveis a essa arquitetura alvo e conclui o sistema alvo mapeando os padrões identificados para os componentes do sistema legado e, por fim, implementa o sistema alvo.	Não	Reengenharia

Outra problemática está relacionada à identificação de objetos em sistemas legados procedimentais, e é comentada por alguns autores (Cha et al., 2004; Cimitile et al., 1999; van Deursen e Kuipers, 1998). De acordo com Cimitile et al. (1999), a identificação dos objetos é realizada com o apoio do engenheiro de software e requer um grande esforço de compreensão. Por outro lado, diversos métodos de reengenharia tentam identificar objetos automaticamente por meio de análise estática do código fonte. No entanto, como confirmado por Cha et al. (2004), a semântica do negócio é perdida. O uso de linguagens de padrões de análise na reengenharia é utilizado nesta tese tentando minimizar essa problemática, pois o entendimento e a identificação dos requisitos do sistema legado são apoiados pelo conhecimento do domínio embutido na linguagem de padrões de análise.

2.3 Padrões e Linguagens de Padrões

2.3.1 Conceitos

A existência de diversas soluções para problemas similares de software fomentou a iniciativa, no início da década de 90, de organizá-las em padrões de software e divulgá-las para serem utilizadas por outras pessoas a fim de poupar tempo e esforço no desenvolvimento de software. Estabeleceu-se uma padronização quanto ao estilo e à forma de apresentação dessas soluções (Appleton, 1997; Beck e Johnson, 1994; Gamma et al., 1995) para permitir a sua evolução e divulgação, além de facilitar a comunicação entre os desenvolvedores que as utilizam. A idéia de padrões de software foi embasada no conceito de padrões, originalmente aplicados na área de Engenharia Civil por Alexander et al. (1977) para documentar conhecimento relacionado ao projeto e à construção de casas e prédios. De acordo com esse autor, "cada padrão descreve um problema que ocorre diversas vezes em nosso ambiente, e então descreve o núcleo da solução para esse problema, de forma que você possa utilizar essa solução milhares de vezes sem usá-la do mesmo modo duas vezes".

Similarmente aos padrões de Alexander et al. (1977), padrões de software são decisões recorrentes tomadas por desenvolvedores experientes e registradas para que os menos experientes possam utilizá-las (Beck e Johnson, 1994), isto é, são unidades de experiência efetivamente transferíveis e uma fonte de discussão e de esclarecimento (Stevens e Pooley, 1998).

Estudos mostram que, quando especialistas trabalham em um problema particular, é raro que inventem uma nova solução completamente diferente das já existentes para resolvê-lo. Diferentes soluções de projeto são conhecidas por eles, de acordo com a própria experiência ou a de outros profissionais. Quando se confrontam com novos problemas, freqüentemente lembram-se de outros similares e reusam a solução antiga, pensando em pares "problema/solução". Esses pares podem ser agrupados em famílias de problemas e soluções similares, sendo que cada família exibe um padrão tanto de problema quanto de solução (Buschmann et al., 1996).

Padrões de software são criados sempre que um conjunto de princípios gerais e de soluções idiomáticas é criado por desenvolvedores experientes para guiar a criação do software e esses princípios e soluções são descritos em um formato estruturado (nome do padrão, problema e solução) e são amplamente adotados por profissionais de software (Larman, 2004a).

Apesar de padrões de software serem utilizados desde a década de 80, tornaram-se populares somente na década de 90 com o livro "Design Patterns: Elements of Reusable Object-Oriented Software" de Gamma et al. (1995). Padrões de software são usados sob diversos níveis de abstração: padrões de análise (Coad, 1992; Coad et al., 1997; Larman, 2004a), de projeto (Gamma et al., 1995; Larman, 2004a), arquiteturais (Beck e Johnson, 1994; Buschmann et al., 1996), de testes (Binder, 1999; DeLano e Rising, 1998), de engenharia

reversa (Demeyer et al., 2000), de reengenharia (Lemos, 2002; Recchia, 2002; Stevens e Pooley, 1998), entre outros.

Padrões de análise descrevem soluções para problemas de análise de sistemas, embutindo conhecimento sobre um domínio de aplicação específico. Padrões arquiteturais descrevem soluções da organização estrutural fundamental de sistemas de software ou hardware (Braga, 2003). Padrões de projeto descrevem soluções para problemas de projeto de software e ensinam os projetistas a padronizarem o modo como desenvolvem. De acordo com Gamma et al. (1995), com o emprego de padrões de projeto no desenvolvimento de software espera-se diminuir os esforços e o tempo de implementação, produzir sistemas flexíveis, reusáveis, com alta qualidade e mais confiáveis, já que as soluções foram testadas e aprovadas em outros sistemas. Além disso, o aumento da manutenibilidade de software pode ser parcialmente alcançado com o uso de padrões de projeto tanto no desenvolvimento quanto na reengenharia de sistemas (Tahvildari et al., 2003). Padrões de teste fornecem diretrizes para auxiliar os testadores na avaliação da qualidade do produto. Padrões de engenharia reversa descrevem como entender e elaborar a documentação a partir de um sistema legado e padrões de reengenharia descrevem soluções de como conduzir a reengenharia e são muito mais amplos do que padrões de projeto, pois incorporam contexto do negócio e do software, fatores de orçamento da organização e o comportamento dos gerentes, cujo apoio é necessário (Stevens e Pooley, 1998).

Além de existirem padrões específicos de engenharia reversa e de reengenharia (Demeyer et al., 2000; Goedicke e Zdun, 2002; Lemos, 2002; Recchia, 2002; Stevens e Pooley, 1998), há também iniciativas do uso de padrões de projeto na reengenharia de sistemas legados (Cha et al., 2004; Chu et al., 2001, 2000; Gall et al., 1996; Tahvildari e Kontogiannis, 2002; Tahvildari et al., 2003), ((Arsanjani, 2001a) **apud** (Cha et al., 2004)), a fim de apoiar a recuperação e a construção da arquitetura do sistema alvo, tornando-o mais flexível e reusável. Além disso, padrões de projeto padronizam e tornam o projeto mais compreensível, colaborando para que a evolução do sistema seja mais efetiva (Chu et al., 2001).

Ressalta-se que a maioria dos trabalhos que tratam da reengenharia baseada em padrões estão mais preocupados com o entendimento e a recuperação da estrutura e da arquitetura do sistema legado. O entendimento da funcionalidade depende totalmente da experiência do engenheiro de software no domínio do sistema legado. Isso pode comprometer a reengenharia quando não houver esse tipo de profissional disponível. A partir disso e observando a vantagem dos padrões de análise de fornecer conhecimento sobre um domínio de aplicação específico, notou-se a possibilidade de utilizar nesta tese linguagem de padrões de análise na reengenharia de sistemas legados para alcançar reúso não somente das soluções de análise (documentação) como também do conhecimento do domínio.

Appleton (1997) afirma que se um padrão é uma solução para um problema em um determinado contexto, então uma linguagem de padrões é um coletivo de tais soluções que agem juntas para resolver um problema, de acordo com um objetivo pré-definido. Uma

linguagem de padrões pode ser considerada também como uma coleção estruturada de padrões que transforma necessidades e restrições em arquitetura (Coplien, 1998). Em geral, a linguagem de padrões é utilizada desde o início e diversos padrões podem ser aplicados várias vezes para resolver um problema (Roberts e Johnson, 1998). Como uma linguagem de padrões possui vários padrões, grande parte da linguagem é formada pelos padrões em si. Outra parte consiste da visão geral do domínio coberto pela linguagem, de sua aplicabilidade e de uma visão geral da interação entre os padrões (Braga, 2003).

Uma linguagem de padrões inclui regras e diretrizes que explicam como e quando aplicar os seus padrões para resolver um problema que é maior do que qualquer padrão individual pode resolver (Appleton, 1997). Além disso, fornece uma solução completa para um conjunto de problemas repetidos, necessitando ser evoluída periodicamente para que as variações de suas soluções sejam atualizadas (Arsanjani, 2001b), o que determina a evolução de frameworks baseados em linguagens de padrões.

Existem diversas iniciativas de linguagens de padrões para serem aplicadas em diferentes contextos. Nesta tese será dado enfoque em linguagens de padrões de análise no domínio de negócio, como é o caso das linguagens de padrões de Ramesh (1998), Braga et al. (1999), Arsanjani (2001b), Ré et al. (2001), Ré (2002), Pazin (2004), e Pazin et al. (2004). Nas de Ramesh (1998) e Arsanjani (2001b) há preocupação apenas com a análise, projeto e implementação de regras do negócio, enquanto que nas de Braga et al. (1999), Ré et al. (2001), Ré (2002), Pazin (2004) e Pazin et al. (2004) considera-se a análise de regras do negócio, bem como a análise da funcionalidade de subdomínios específicos do negócio. O interesse desta tese está concentrado nas linguagens de padrões desse segundo grupo.

Uma outra carência observada durante o levantamento bibliográfico realizado foi a ausência de iniciativas de testes associados a padrões de software, nos diferentes níveis de abstração (por exemplo, análise, projeto e implementação). Isso é uma preocupação eminente, já que padrões são utilizados no desenvolvimento, manutenção e reengenharia de software, e atividades de teste fazem parte da garantia de qualidade de qualquer produto desenvolvido (Rocha et al., 2001). Neste trabalho, teste será considerado sob a perspectiva de padrões de linguagem de padrões de análise, visando propiciar o reúso de teste funcional na reengenharia e no desenvolvimento de software quando se utiliza esse tipo de linguagem. Isso será discutido nos Capítulos 3 e 5.

2.3.2 Linguagens de Padrões de Análise Relevantes no Domínio de Sistemas de Informação

Embora inúmeras linguagens de padrões tenham surgido na última década, poucas são as que podem ser adequadamente aplicadas ao contexto de negócios. Nesta seção apresenta-se em mais detalhes a linguagem de padrões GRN, proposta por Braga et al. (1999), e comentam-se outras linguagens e padrões de análise encontrados na literatura.

2.3.2.1 Uma Linguagem de Padrões para Gestão de Recursos de Negócios - GRN

Braga et al. (1999) propuseram uma linguagem de padrões para Gestão de Recursos de Negócios (GRN), que é formada por quinze padrões de análise, alguns dos quais são aplicações ou extensões de padrões existentes na literatura. Entretanto, a linguagem de padrões GRN possui abstração superior em relação a esses padrões da literatura, pois é aplicável a um domínio mais específico e contém valor semântico inerente a uma família de aplicações desse domínio.

A linguagem de padrões GRN oferece aos desenvolvedores inexperientes, informação suficiente para o desenvolvimento de novos sistemas do domínio de Gestão de Recursos de Negócios, juntamente com soluções alternativas.

GRN possui um domínio específico e bem definido, concentrado no aluguel, comércio e manutenção de recursos de negócios. Exemplificando, o aluguel de recursos concentra-se na utilização temporária de um bem ou serviço, por exemplo, a locação de um carro ou o tempo de um especialista. O comércio de recursos concentra-se na transferência de propriedade de um bem, por exemplo, uma venda ou uma compra de produtos. A manutenção de recursos concentra-se na manutenção de um determinado bem, utilizando mão-de-obra e peças para execução, por exemplo, o reparo de eletrodomésticos em uma oficina auto-elétrica especializada.

Na Figura 2.2 apresentam-se os padrões da linguagem GRN, as dependências existentes entre eles e a ordem em que eles podem ser aplicados. Essa linguagem possui três padrões principais: LOCAR O RECURSO (4), COMERCIALIZAR O RECURSO (6) e MANTER O RECURSO (9). A aplicação de cada um desses padrões e, conseqüentemente, dos que a esses estão relacionados, é realizada de acordo com o objetivo do sistema que se deseja modelar. O uso desses padrões não é mutuamente exclusivo, uma vez que existem sistemas nos quais eles podem ser usados concomitantemente, por exemplo, em uma oficina mecânica de veículos que compra e vende peças e também conserta carros.

O primeiro padrão que deve ser aplicado quando da utilização da linguagem é IDENTIFICAR O RECURSO (1). Os padrões ITEMIZAR TRANSAÇÃO DO RECURSO (11), PAGAR PELA TRANSAÇÃO DO RECURSO (12) e IDENTIFICAR O EXECUTOR DA TRANSAÇÃO (13) são exibidos dentro de um retângulo, o que mostra que esses padrões são aplicáveis a todas as situações nas quais uma seta chega até a sua borda. Por exemplo, a seta sem origem que chega ao padrão 11 significa que esse é o primeiro padrão a ser aplicado, seguido opcionalmente dos padrões 12 e 13.

Como apresentado na Figura 2.2, os padrões da linguagem estão reunidos em três grupos, de acordo com seu objetivo. O grupo 1 (Identificação de Recurso de Negócio) possui três padrões, (1), (2) e (3), que dizem respeito à identificação e possível qualificação, quantificação e armazenamento dos recursos gerenciados pelo negócio. O grupo 2 (Transações de Negócio), possui sete padrões, (4) a (10), que estão relacionados à manipulação dos recursos do negócio

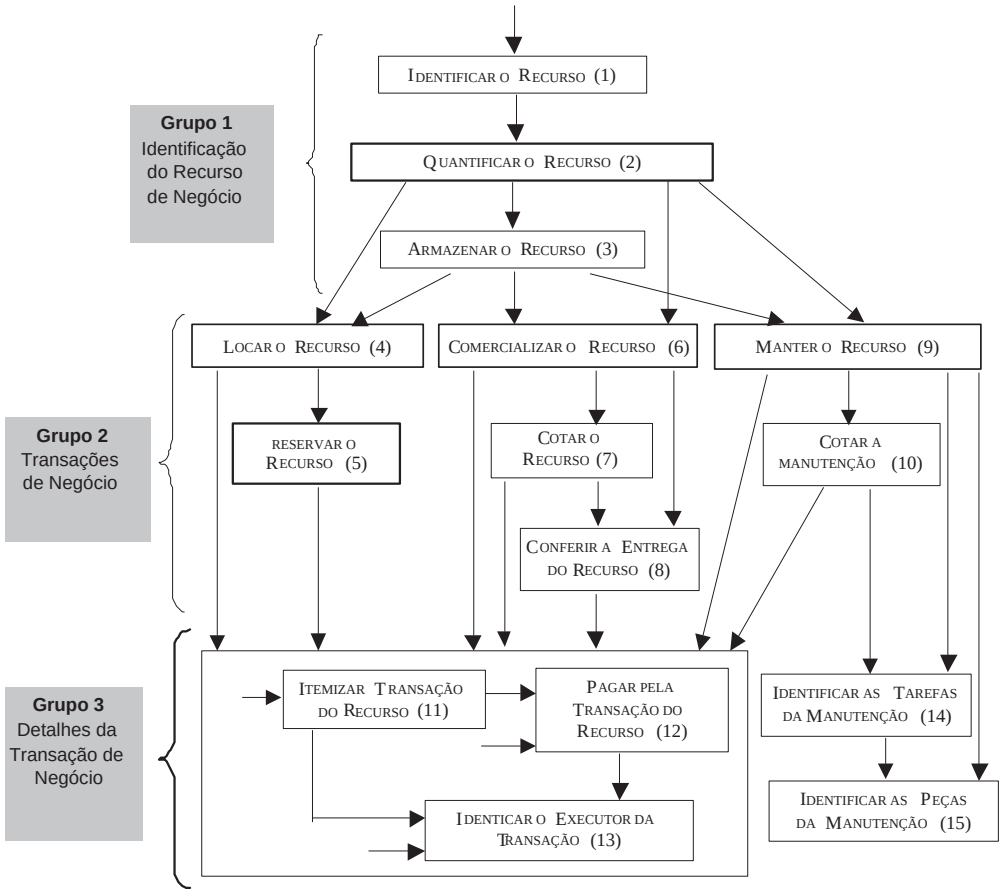


Figura 2.2: Estrutura da linguagem de padrões GRN (Braga et al., 1999)

pela aplicação. O grupo 3 (Detalhes da Transação de Negócio) contém cinco padrões, (11) a (15), que tratam os detalhes das transações efetuadas com o recurso.

A estrutura dos padrões é representada em UML (*Unified Modelling Language*) (Fowler e Scott, 1997; OMG, 2003) e novos atributos podem ser adicionados às classes, de acordo com a necessidade do sistema, tornando a GRN flexível. Além de métodos, GRN representa também operações do sistema, que são consideradas mais que métodos, pois são executadas em resposta a eventos que ocorrem no mundo real.

- **Usos e Extensões da GRN**

A linguagem de padrões LV (Linguagem de Padrões para **L**eilões **V**irtuais), proposta por Ré et al. (2001) e Ré (2002), tem como propósito apoiar o desenvolvimento de sistemas para gestão de vendas por meio de leilões virtuais. Sua construção baseou-se em três sistemas de leilões encontrados na Internet. Os padrões da LV são agrupados em duas categorias: 1) padrões requeridos e 2) padrões opcionais. A primeira categoria contém os padrões essenciais para leiloar requisitos e a segunda contém padrões que são desejáveis e não precisam ser obrigatoriamente utilizados. Essa linguagem pode ser considerada uma extensão da GRN, pois trata de uma transação (ou seja, leilão) não coberta pela GRN.

Pazin (2004) e Pazin et al. (2004) propuseram uma linguagem de padrões no domínio de clínicas de reabilitação, denominada SiGCli (Sistemas de Gerenciamento de Clínicas), que trata do controle de atendimentos a pacientes, do controle de vendas (de serviços e de produtos) e de compras, bem como o controle do faturamento (contas a pagar e a receber). Por exemplo, em uma clínica de fisioterapia, controla-se o atendimento dos pacientes pelo fisioterapeuta responsável; vendem-se os serviços dos fisioterapeutas (por exemplo, RPG - re-educação postural global - e massagem para problemas respiratórios) ou produtos (por exemplo, produtos fisioterápicos); compram-se produtos que são utilizados durante as sessões de fisioterapia ou para a manutenção da clínica; e controlam-se as contas a pagar e as contas a receber. Apesar dessa linguagem de padrões ter nome diferente, ela não difere na essência da GRN, pois os seus padrões foram baseados nos padrões da GRN, sendo que em alguns foi mantida a estrutura original. O único padrão originalmente definido na SiGCli trata do acompanhamento dos recursos, e que não foi previsto na GRN. No contexto de clínicas de reabilitação esse padrão controla o histórico da evolução dos pacientes, que permite realizar avaliações freqüentes dos pacientes mediante dados clínicos coletados. Assim, a SiGCli pode ser considerada como uma extensão da GRN e não como uma nova linguagem de padrões.

2.3.2.2 Outros Padrões e Linguagens de Padrões de Análise

"Cenário da Aplicação" (*Application Scenario*) é uma linguagem de padrões, proposta por Ramesh (1998), indicada para sistemas em que os processos, as regras e as interações entre os objetos de negócios são a parte mais importante do sistema. Nessa linguagem são descritos diversos padrões que: 1) tornam os processos do negócio explícitos e 2) modelam e implementam esses processos desde o levantamento de requisitos e análise até o projeto e a implementação dos objetos.

Arsanjani (2001b) afirma que regras do negócio mudam todos os dias devido à volatilidade dos requisitos do negócio. Além disso, tendem a mudar com mais freqüência do que o resto dos objetos do negócio com que elas estão associadas. Essas regras são geralmente implementadas nos métodos de um objeto do negócio, mas também se referem a outros objetos do negócio que estão relacionados a elas indiretamente. Isso cria um emaranhado de dependências implícitas, diminuindo ainda mais a manutenibilidade do sistema. Baseado nessas tendências, Arsanjani (2001b) criou a linguagem de padrões "Objeto Regra" (*Rule Object*) para solucionar os seguintes problemas: Como é possível manipular mudanças e manter os sistemas passíveis de manutenção, reusáveis e com capacidade de extensão? Como modelar, manipular e representar regras para aumentar o reúso, a manutenibilidade e o desempenho dos sistemas?

O padrão **"Objeto Regra"**, considerado por Arsanjani (2001b) o mais importante da sua linguagem de padrões, tem como objetivo separar os elementos das regras que podem mudar com mais freqüência daqueles que não podem. Esse padrão foi utilizado em alguns projetos:

na camada Objetos de Negócio Comuns da arquitetura do framework IBM SanFrancisco (Subseção 2.4.2.2 deste capítulo); no framework *"If-Then-Else"* [2], que é um framework de pequeno porte e tem como objetivo escrever código (Java) com ramificação lógica eliminando *"if's"* aninhados; e no servidor de aplicações *IBM WebSphere Application Server Enterprise Edition* [3], que fornece uma infra-estrutura de software e permite a criação de aplicações *e-business* adaptáveis e flexíveis. Arsanjani (2001b) não apresenta a limitação do domínio de negócios em que a sua linguagem de padrões pode ser aplicada. Uma das idéias básicas desta linguagem de padrões é que as regras não são simplesmente uma questão técnica, elas são principalmente uma questão de negócio que devem ser organizadas e gerenciadas com atenção e precaução.

Fernandez e Liu (2002) apresentam um padrão de análise que trata do controle de contas (criação, fechamento, transferência e saldo) dos clientes de uma determinada corporação, bem como das diversas transações que são realizadas por eles em suas respectivas contas. Um outro padrão de análise que controla contas foi encontrado na literatura (Fayad e Hamza, 2003); no entanto, é mais genérico do que o de Fernandez e Liu (2002), pois modela qualquer tipo de conta, com qualquer tipo de regra ou regulamento, controlado por qualquer parte interessada (uma pessoa, uma empresa, um grupo de pessoas ou de empresas). Esse padrão pode ser aplicado em qualquer aplicação, independentemente do domínio a que pertence.

No mesmo contexto da linguagem de padrões SiGCli, Sorgente et al. (2004) apresentam padrões de análise que fornecem soluções de alguns aspectos relacionados ao tratamento de um paciente em um hospital, como: gerenciamento do prontuário do paciente, controle dos bens e serviços designados para cada tratamento e o controle do histórico dos tratamentos realizados.

2.3.3 Resumo das Linguagens de Padrões Estudadas

A partir do estudo das linguagens de padrões apresentadas nas Subseções 2.3.2.1 e 2.3.2.2, notou-se que, apesar das cinco linguagens estarem no domínio de negócios, elas se concentram em níveis diferentes de abrangência e abstração. Nas linguagens de padrões "Objeto Regra" e "Cenário da Aplicação" há preocupação com a modelagem, projeto e implementação das regras do negócio de sistemas em domínio não especificado, enquanto que nas linguagens de padrões GRN, SiGCli e LV há preocupação com a análise das regras do negócio e da funcionalidade de sistemas em um domínio específico e bem definido. Essa característica das linguagens de padrões GRN, SiGCli e LV pode ser uma vantagem em relação às outras quando o interesse está em utilizar linguagem de padrões de análise para apoiar o entendimento e a elaboração da documentação de sistemas legados na reengenharia. Além disso, foi possível identificar que nas linguagens de padrões "Objeto Regra" e "Cenário da Aplicação" classes

[2] `http://www.javaworld.com/javaworld/jw-03-2000/jw-0324-ifthenelse.html`
[3] `http://www-3.ibm.com/software/webservers/appserv/enterprise.html`

são criadas para gerenciar as regras do negócio, enquanto que nas linguagens de padrões GRN, SiGCli e LV esse gerenciamento é feito nas próprias classes do sistema.

Na Tabela 2.2 apresentam-se alguns aspectos observados durante o estudo das linguagens de padrões discutidas.

Tabela 2.2: Resumo das linguagens de padrões de análise estudadas

Linguagens de Padrões/ Características	**"Objeto Regra"** (Arsanjani, 2001b)	**"Cenário da Aplicação"** (Ramesh, 1998)	**GRN** (Braga et al., 1999)	**SiGCli** (Pazin, 2004; Pazin et al., 2004)	**LV** (Ré, 2002; Ré et al., 2001)
Nível de Abstração	Análise, Projeto e Implementação	Análise, Projeto e Implementação	Análise	Análise	Análise
Abrangência	Domínio não especificado	Domínio não especificado	Domínio de Gestão de Recursos de Negócios	Domínio de Gestão de Recursos de Negócios	Domínio de Leilões Virtuais
Local de Gerenciamento das Regras	Objetos Específicos	Objetos Específicos	Objetos do Sistema	Objetos do Sistema	Objetos do Sistema
Utilizada por algum framework ou gerador?	Frameworks SanFrancisco e *"if-then-else"*; e *IBM WebSphere Application Server Enterprise Edition*	Não	Framework GREN (Braga, 2003)	Gerador GAwCRe (Pazin, 2004)	Framework Qd+ (Ré, 2002)

A GRN foi selecionada para ser utilizada nos estudos de caso de reengenharia utilizando o processo de reengenharia definido nesta tese (apresentado no Capítulo 3) pois: atende ao domínio de Gestão de Recursos de Negócios, que é um subdomínio do domínio de Sistemas de Informação, que é de interesse desta tese, como discutido no Capítulo 1; e existe um framework cuja construção foi baseada nela e foi desenvolvida no mesmo Grupo de Pesquisa deste trabalho.

2.4 Frameworks

2.4.1 Conceitos

Muitas definições para frameworks de software são encontradas na literatura. Contudo, de acordo com Braga (2003), todas têm em comum o enfoque na sua característica mais marcante: facilitar o reúso. Entende-se por reúso não só o aproveitamento de trechos de código de programas, mas também o reúso de esforços realizados em todas as fases do desenvolvimento de software.

O maior benefício de utilizar frameworks é que eles permitem um nível maior de reúso de código e projeto do que aquele que é alcançado com outras abordagens de projeto. Os benefícios dos frameworks e do reúso são obtidos com o tempo, uma vez que a produtividade não é alcançada no primeiro ou no segundo uso, mas em diversos usos da tecnologia (Brugali et al., 2000; Taligent, 1998). Algumas vantagens que podem ser obtidas da utilização de frameworks são (Bosch et al., 1999; Brugali et al., 2000; Fayad e Johnson, 2000; Pree, 1999; Taligent, 1998): obtenção de diretrizes de arquitetura e de infra-estrutura e também da especialidade do domínio; reúso de funções implementadas em um determinado domínio; e redução da manutenção e do tempo de desenvolvimento. Isso motivou o uso de frameworks nesta tese como apoio computacional na reengenharia de software.

Um framework de software é uma arquitetura reusável que fornece comportamento e estrutura genérica para uma família de abstrações de software (Appleton, 1997). No entanto, Fayad e Johnson (2000) afirmam que um framework é mais do que uma arquitetura, é uma aplicação semi-completa contendo componentes estáticos e dinâmicos que podem ser adaptados para produzir aplicações específicas do usuário. Larsen (1999) define framework como um padrão arquitetural, que fornece um gabarito extensível para criar aplicações dentro de um domínio.

Um framework é considerado também como um conjunto de blocos de construção de software pré-fabricados que programadores podem usar, estender ou ajustar para soluções específicas de computação. Com frameworks, programadores não têm que começar do nada toda vez que forem escrever uma aplicação. Além disso, um framework mantém a especialidade de programação, necessária para resolver uma classe particular de problemas (Taligent, 1997).

Frameworks possuem em sua arquitetura partes fixas (*frozen spots*) e partes variáveis (*hot spots*) (Buschmann et al., 1996). As partes fixas definem a arquitetura geral de uma aplicação - seus componentes básicos e os relacionamentos entre eles - e são usadas sem nenhuma modificação em todas as instanciações de um framework. As partes variáveis, em geral, implementadas usando *callbacks* ou polimorfismo, contém componentes que podem ser adaptados e estendidos para satisfazer as necessidades de um sistema específico (por exemplo, regras do negócio e requisitos funcionais inerentes às políticas internas da empresa).

Mais especificamente, os *hot spots* são classes ou métodos abstratos do framework que devem ser implementados com código específico da aplicação durante a instanciação do framework. Ressalta-se que frameworks não são executáveis; para gerar um executável é necessário instanciá-lo (Markiewicz e Lucena, 2001).

Uma outra denominação semelhante aos *hot spots* é a de gancho (*hook*), que consiste de um ponto do framework passível de ser adaptado de alguma forma, por exemplo, por meio do preenchimento de parâmetros ou da criação de subclasses ((Froehlich et al., 1997) **apud** (Braga, 2003)). A descrição de cada gancho documenta um problema ou um requisito do

framework e oferece diretrizes de como utilizar o gancho para satisfazer tal requisito (Braga, 2003).

Existem muitos tipos de frameworks no mercado para resolver vários tipos de problemas e nem todos são orientados a objetos. Os tipos de frameworks vão desde frameworks de aplicação, que ajudam no desenvolvimento de aplicações específicas, até frameworks de nível de infra-estrutura, que fornecem os serviços de software básico tal como suporte de comunicação, de impressão e de sistemas de arquivos. Existem também frameworks de domínio específico que se destinam a problemas em áreas particulares. Fayad e Schmidt (1997) classificam os frameworks em três grupos, de acordo com o escopo a que são destinados: frameworks de infra-estrutura do sistema, frameworks de integração *middleware* e frameworks de aplicação empresarial. Os frameworks de aplicação empresarial (de interesse desta tese) permitem o desenvolvimento direto de aplicações e produtos, poupando tempo e esforço humano em domínios de aplicação mais amplos e são muito úteis para atividades de negócios das empresas. São mais caros para desenvolver ou comprar, mas podem dar um melhor retorno em relação ao investimento realizado. O Framework IBM SanFrancisco, o framework Qd+ (ambos comentados na Subseção 2.4.2.2) e o framework GREN (Subseção 2.4.2.1) são alguns exemplos desse tipo de framework.

Outra classificação de frameworks, abordada em Taligent (1997), é baseada em como um framework é utilizado, ou seja, se o desenvolvedor deriva novas classes ou instancia e combina as classes existentes. Essa distinção é algumas vezes referida como guiada a arquitetura (*architecture-driven*) ou concentrada em herança (*inheritance-focused*) e guiada a dados (*data-driven*) ou concentrada em composição (*composition-focused*).

Além da classificação quanto ao escopo e quanto à forma de utilização, frameworks também podem ser classificados pelas técnicas que são usadas para estendê-los (Fayad e Johnson, 2000). Essa classificação abrange desde frameworks caixa-branca (*white box*) até frameworks caixa-preta (*black box*). Frameworks caixa-branca baseiam-se em características de linguagens orientadas a objetos, como herança e ligação dinâmica, para alcançar a capacidade de extensão. O framework caixa-branca pode ser difícil de aprender para ser utilizado, uma vez que o estudo necessário para utilizá-lo é o mesmo que aquele requerido para saber como ele é construído (Johnson e Foote, 1988). A funcionalidade existente é reutilizada e estendida por meio de herança a partir das classes base do framework e a partir da sobreposição de métodos pré-definidos.

Frameworks caixa-preta apóiam capacidade de extensão, definindo interfaces para componentes que podem ser conectados ao framework por meio da composição de objetos. A funcionalidade existente é reutilizada por meio da definição de componentes que obedecem a uma interface particular e a partir da integração desses componentes ao framework usando padrões, como aqueles definidos por Gamma et al. (1995). Além disso, frameworks caixa-preta são mais fáceis de aprender e usar do que os caixa-branca, mas são menos flexíveis (Johnson e Foote, 1988). Existem também frameworks caixa-cinza, que contém

características tanto de frameworks caixa-branca quanto de frameworks caixa-preta. A capacidade de extensão é dada por meio de herança e ligação dinâmica, assim como pela definição de interfaces.

Na Tabela 2.3 são apresentadas resumidamente as diversas classificações de frameworks, discutidas anteriormente. Na primeira coluna do quadro relata-se o nome dos autores que criaram cada classificação, na segunda apresenta-se a razão na qual a classificação é baseada e na última coluna mostram-se os tipos de cada classificação.

Tabela 2.3: Resumo das classificações de frameworks

Autores	**Classificação baseada:**	**Tipos de Classificação**
Fayad e Schmidt (1997)	no escopo a que o framework é destinado.	– frameworks de infra-estrutura do sistema; – frameworks de integração *middleware*; – frameworks de aplicação empresarial.
Taligent (1997)	em como um framework é utilizado.	– utilização guiada a arquitetura ou concentrada em herança; – utilização guiada a dados ou concentrada em composição.
Fayad e Johnson (2000)	nas técnicas que são usadas para estender o framework.	– frameworks caixa-branca; – frameworks caixa-preta; – frameworks caixa-cinza.

Outra tendência de pesquisa em frameworks está relacionada ao apoio de ferramentas para apoiar tanto a evolução de frameworks quanto a dos sistemas criados a partir da instanciação deles. Tourwé (2002) propõe uma ferramenta que fornece transformações automáticas de alto nível para instanciar e evoluir corretamente frameworks. Essa ferramenta detecta e evita desvios (*drifts*) na implementação de frameworks, avaliando o impacto causado pela evolução do framework e de seus sistemas existentes.

Apesar da ausência na literatura de processos de evolução específicos para frameworks, uma técnica denominada COSVAM (***CO****VAMOF* [4] ***S****oftware* ***V****ariability* ***A****ssessment* ***M****ethod*) (Deelstra et al., 2004), que avalia a variabilidade de família de produtos [5] ((Clements e Northrop, 2001) **apud** (Deelstra et al., 2004)) para apoiar sua evolução foi encontrada. Essa técnica consiste de cinco passos: 1) determinar o objetivo da avaliação de variabilidade, 2) especificar a variabilidade fornecida, 3) especificar a variabilidade requerida, 4) avaliar

[4]Técnica de modelagem de variabilidade que representa tanto a variabilidade requerida como a variabilidade fornecida pela família de produtos e usa as representações para avaliar o desacordo entre esses dois tipos de variabilidades (Sinnema et al., 2004b). COVAMOF apóia o desenvolvimento e a evolução da família de produtos e a derivação de produtos a partir da família de produtos (Sinnema et al., 2004a).

[5]Famílias de produtos, geradas por linhas de produtos de software (*software product-lines*) (Weiss e Lai, 1999), consiste de uma arquitetura compartilhada, um conjunto de componentes e um conjunto de produtos. Durante a derivação do produto, cada membro da família de produto deriva sua arquitetura a partir da arquitetura da família de produtos, seleciona e configura um subconjunto dos componentes da família de produtos. Variabilidade na família de produtos é fornecida em termos de pontos de variação (ou *hot spots*) com mecanismos associados, dependências e restrições (Deelstra et al., 2004). Como frameworks, famílias de produtos de software têm sido uma abordagem efetiva de reúso no desenvolvimento de software (Sinnema et al., 2004a).

o desacordo entre a variabilidade fornecida e a requerida e 5) interpretar os resultados da avaliação.

De acordo com Deelstra et al. (2004), o primeiro passo pode assumir um dos cinco objetivos que podem ser apoiados pela técnica: 1) determinar a habilidade da família de produtos derivar um novo produto; 2) determinar se as características não fornecidas, durante a derivação de produtos, devem ser implementadas no produto ou ser integradas à família de produtos; 3) coletar dados para o planejamento da liberação de nova versão da família de produto; 4) avaliar o impacto de novas características que estão presentes nos produtos da família; 5) determinar se toda a variabilidade fornecida é ainda necessária.

Apesar da técnica COSVAM fornecer suporte para o gerenciamento de variabilidade, não há um processo de evolução específico que pode ser seguido quando a decisão é evoluir a família de produtos. Isso também ocorre no contexto de frameworks e motivou, nesta tese, a definição de um processo de evolução de frameworks de aplicação, apresentado no Capítulo 4.

Como comentado nesta seção, há diversas iniciativas de frameworks para ser utilizados em diversos escopos, seja em níveis de menor abstração, por exemplo em sistemas operacionais, como em níveis de maior abstração, como em sistemas comerciais. Essa tecnologia está evoluindo cada vez mais devido ao alto nível de reúso que proporciona e, conseqüentemente, ao aumento da produtividade nas indústrias de software (Markiewicz e Lucena, 2001). Por essa razão, juntamente com a necessidade e a importância de apoio computacional na reengenharia de software, o uso de frameworks de aplicação empresarial na reengenharia de sistemas é explorado nesta tese, como já mencionado no início desta seção. Alguns frameworks do domínio de interesse são abordados na Subseção 2.4.2.

2.4.2 Frameworks de Aplicação Relevantes no Domínio de Sistemas de Informação

Nesta subseção apresentam-se alguns frameworks do domínio de negócios, que é de interesse desta tese. A maioria deles faz parte do conjunto de frameworks de aplicação empresarial. O framework GREN, proposto por Braga (2003), é apresentado em mais detalhes.

2.4.2.1 Framework GREN

O framework GREN (Braga, 2003) pertence ao domínio de Gestão de Recursos de Negócios e sua construção foi baseada na linguagem de padrões GRN, apresentada na Subseção 2.3.2.1. Como é baseado na GRN, as classes e os relacionamentos contidos em cada padrão dessa linguagem de padrões possuem implementação correspondente na hierarquia de classes do GREN. As variantes de cada padrão foram implementadas como *hot spots* no framework. Assim, evoluções na GRN podem causar evoluções no framework GREN e evoluções no framework podem causar também evoluções na GRN. O processo de evolução de frameworks

de aplicação, proposto nesta tese e apresentado no Capítulo 4, pode ser usado para apoiar a evolução do GREN.

O objetivo do GREN é apoiar o desenvolvimento de aplicações no domínio de Gestão de Recursos de Negócios. Dentre essas aplicações, podem ser citadas aquelas que tratam aluguel, comércio ou manutenção de recursos, por exemplo, locadoras de vídeo, bibliotecas, hotéis, lojas, oficinas de consertos de eletrodomésticos e oficinas mecânicas.

A linguagem de programação utilizada na implementação do GREN foi Smalltalk (Beck, 1997; Shafer, 1991), com uso do ambiente de desenvolvimento VisualWorks Non-Commercial versão 5i.2 (Cincom, 2003). O armazenamento dos objetos é feito no banco de dados relacional MySQL (MySQL, 2003). O framework GREN é do tipo caixa cinza, pois foi projetado para ser utilizado tanto por herança quanto por composição.

A arquitetura do GREN foi projetada em três camadas: a de persistência, a de negócios e a de interface gráfica com o usuário (GUI), conforme apresentado na Figura 2.3. A **camada de persistência** trata da conexão com a base de dados, do gerenciamento dos identificadores dos objetos e da persistência dos objetos em Smalltalk para o banco de dados relacional MySQL. A **camada de negócios** trata, em geral, da implementação das classes e relacionamentos de cada padrão da linguagem de padrões GRN e comunica-se com a camada de persistência para o armazenamento dos objetos da aplicação criada pelo framework. A **camada de interface gráfica com o usuário** trata da entrada e da saída de dados, permitindo a interação do usuário final com a aplicação. Esta camada comunica-se com as duas camadas diretamente inferiores (ou seja, de negócios e de persistência) para obter dados, representados como objetos, a serem exibidos aos usuário e para entregar dados para serem processados pela camada de negócios. A camada GREN-Wizard, que gera o código da aplicação a partir da sua especificação baseada nos padrões da GRN, situa-se acima da camada do GREN, pois utiliza todas as demais camadas. É possível também criar aplicações, por meio de herança ou por composição de objetos, utilizando as três camadas do GREN ou utilizando apenas as camadas de negócio e de persistência. O último caso ocorre quando não há interesse de reutilizar a interface gráfica com o usuário do GREN. Na Figura 2.3 ilustram-se também os atores que interagem com cada camada, definindo-a ou utilizando-a.

Para utilizar o framework GREN caixa-branca na criação de novas aplicações, o desenvolvedor deve usar herança para obter as classes específicas de sua aplicação. A criação da aplicação é conduzida pelo uso da linguagem de padrões GRN, cujo resultado é o diagrama de classes da aplicação que será criada. Com base nesse diagrama, utilizam-se classes pré-programadas do framework GREN (por exemplo, a hierarquia de classes da camada de negócios que trata dos recursos, conforme parcialmente apresentada na Figura 2.4) para criar o código da nova aplicação. Para auxiliar a instanciação caixa-branca do GREN há um documento de auxílio, denominado em inglês de *cookbook*, que descreve passo a passo como executá-la (Braga, 2002b). Nesse documento podem ser encontradas informações de quais classes abstratas do framework devem ser especializadas, de acordo com os padrões da

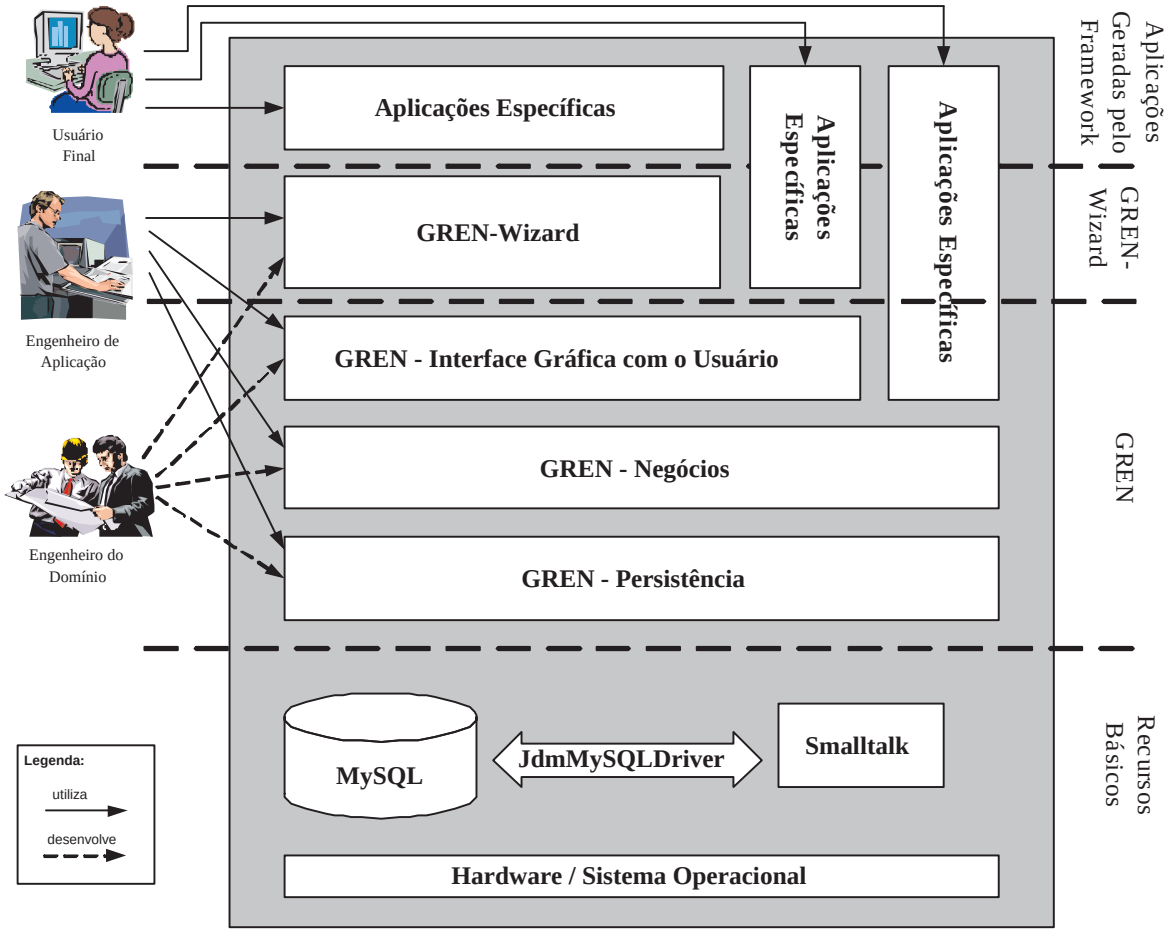


Figura 2.3: Arquitetura do GREN (adaptada de (Braga, 2003))

linguagem GRN que foram utilizados para modelar a aplicação. O uso do GREN caixa-preta é feito por meio de um instanciador [6] (ou ferramenta de instanciação), que está apresentado a seguir.

Ferramenta de Instanciação GREN-Wizard

A ferramenta GREN-Wizard (Braga, 2003; Braga e Masiero, 2003) foi criada para facilitar o uso do framework GREN, uma vez que o entendimento para utilizar frameworks é difícil e consome muito tempo. Essa ferramenta é considerada também como um gerador de aplicações, pois gera classes e métodos a partir da especificação de uma aplicação particular, criada por meio dos padrões da GRN utilizados para modelá-la.

GREN-Wizard visa automatizar os passos indicados pelo documento de auxílio à instanciação. Para isso, o usuário, após ter feito a análise da aplicação usando a linguagem de padrões GRN, responde a diversas perguntas da ferramenta de instanciação e aciona botões para que o código seja gerado automaticamente em Smalltalk e a base de dados seja automaticamente criada no banco de dados MySQL.

[6] Denominado *wizard* em inglês.

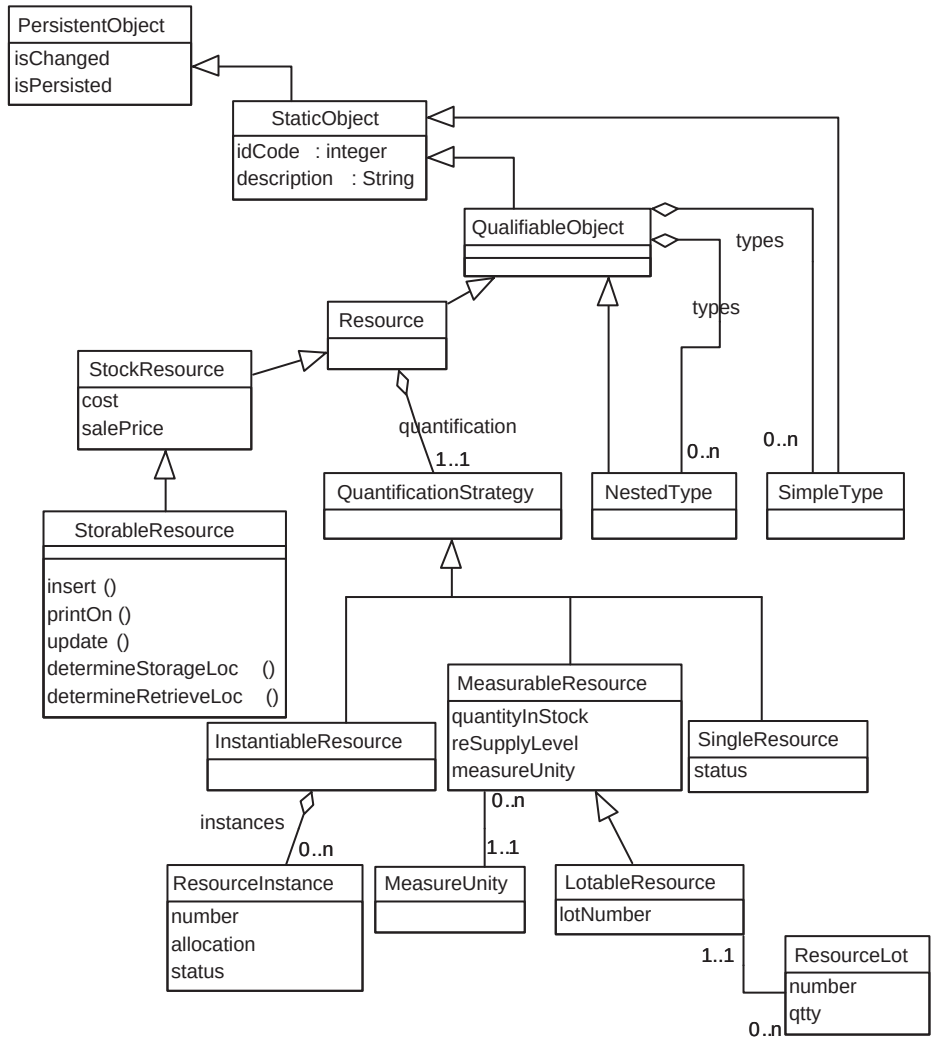


Figura 2.4: Hierarquia parcial de classes do GREN (Braga, 2003)

Na Figura 2.5 apresenta-se a arquitetura da ferramenta GREN-Wizard, que é composta por três módulos: o módulo de especificação do domínio, o módulo de especificação da aplicação e o módulo de geração de código.

No módulo **especificação do domínio**, o engenheiro de domínio especifica a linguagem de padrões GRN e o seu mapeamento para classes do framework GREN, que são armazenadas em uma base de dados MySQL, de acordo com um meta-modelo de dados definido por Braga (2003). Ressalta-se que o engenheiro de domínio é responsável pela construção da ferramenta de instanciação e, provavelmente, também pode ter sido responsável pelo desenvolvimento da linguagem de padrões e por parte da construção do framework. O meta-modelo mapeia informações de cada padrão da GRN, das classes que os compõem, das suas variantes, bem como das classes da camada de interface e de negócio do framework GREN que devem ser consideradas na instanciação do framework quando um determinado padrão e uma determinada variante são utilizados durante a especificação de uma determinada aplicação. No módulo de **especificação da aplicação** o engenheiro de aplicações especifica

a aplicação por meio dos padrões da GRN, usando uma interface gráfica com o usuário (GUI). A especificação da aplicação é armazenada em uma base de dados MySQL de acordo com um meta-modelo de dados definido por Braga (2003). Esse meta-modelo mapeia a correspondência das classes da aplicação com as classes dos padrões e variantes da GRN que foram utilizados na modelagem e também representa os atributos específicos das classes da aplicação. Finalmente, o módulo de **geração de código** utiliza as informações sobre a especificação do domínio e sobre a especificação da aplicação para gerar o código específico da aplicação, que deve juntar-se ao código do framework para produzir a aplicação a ser entregue ao usuário final.

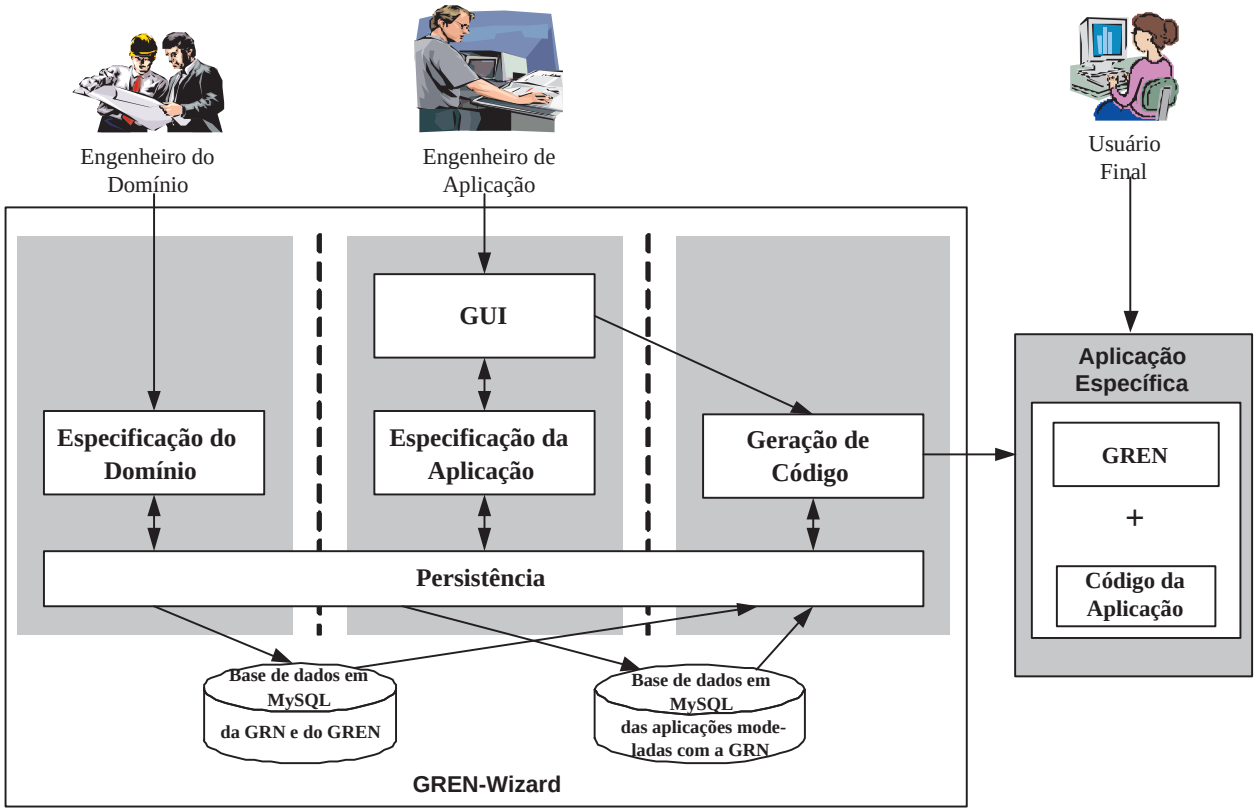


Figura 2.5: Arquitetura da ferramenta GREN-Wizard (Braga, 2003)

Na Figura 2.6 apresenta-se a tela da ferramenta GREN-Wizard, na qual informações sobre a aplicação a ser criada e sobre o uso dos padrões e variantes, no caso específico, do padrão **LOCAR O RECURSO**, são solicitadas. Os botões dessa tela são ativados, de acordo com a necessidade.

O uso da ferramenta GREN-Wizard inicia-se selecionando o nome de uma aplicação já especificada anteriormente (Figura 2.6, #1) ou criando uma nova aplicação (#2), que é incluída na lista de aplicações especificadas. No caso da criação de uma nova aplicação, o nome do primeiro padrão da GRN é apresentado (no mesmo local em que está informado o nome do padrão 4, Figura 2.6, #3), bem como uma lista de suas variantes (#4).

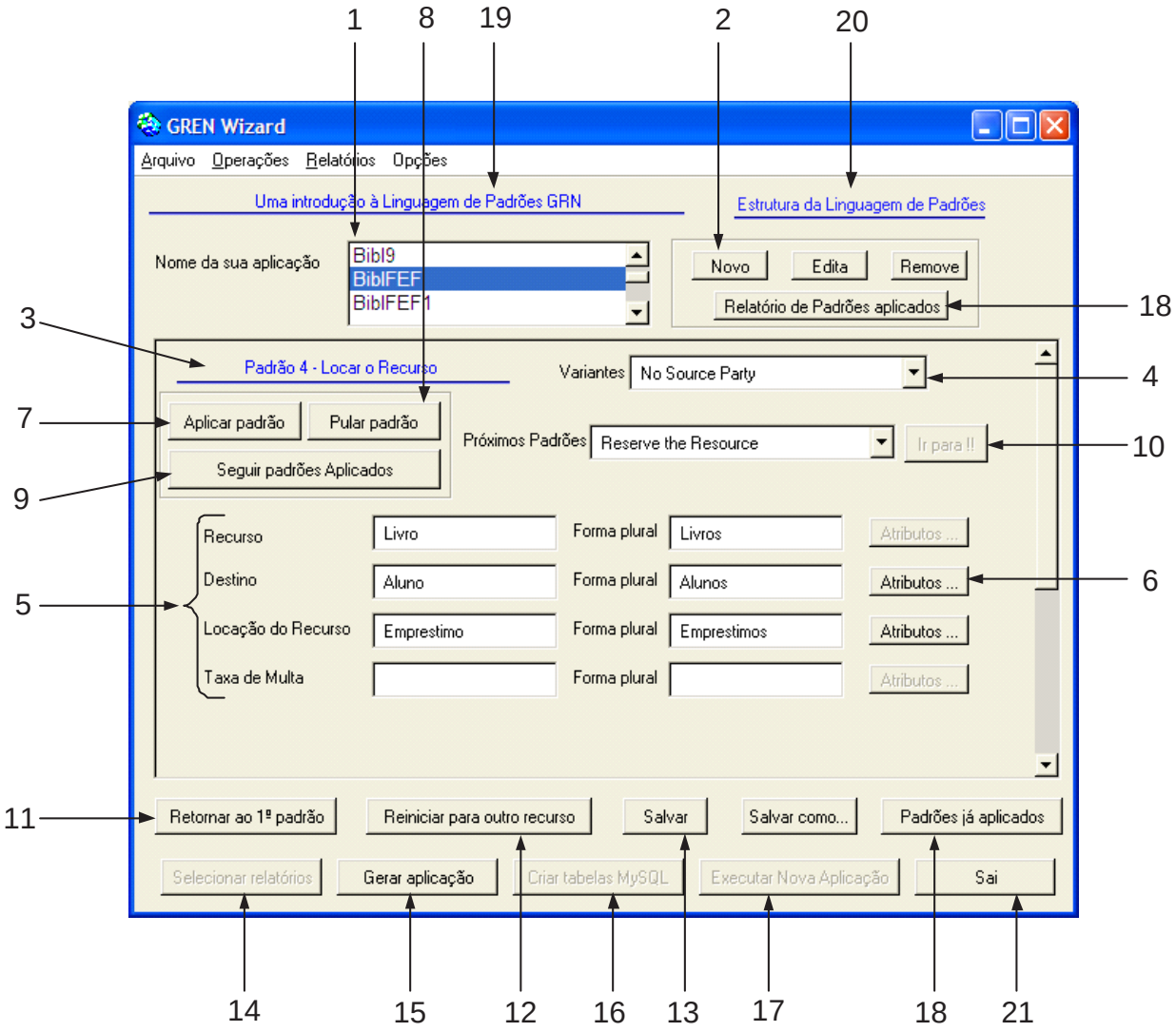


Figura 2.6: Tela da ferramenta GREN-Wizard

As classes participantes da variante do padrão selecionada são apresentadas dinamicamente na interface da ferramenta de instanciação (#5) para que o engenheiro de aplicação especifique as classes da aplicação correspondentes. Novos atributos, específicos das classes da aplicação podem também ser especificados (#6). Após o término da especificação das classes do sistema correspondentes às classes participantes do padrão, é necessário aplicá-lo (#7).

Há também a opção de não aplicar o padrão, ou seja, não utilizá-lo na especificação da aplicação (#8) e também a de seguir os padrões já aplicados (#9). Após a aplicação de um padrão é necessário selecionar o próximo a ser aplicado (#10). A ordem de aplicação dos padrões segue a mesma que a da estrutura da GRN (Figura 2.2, página 21), que está disponível na ferramenta de instanciação (#20).

Em qualquer momento é possível retornar ao início da especificação da aplicação (#11) ou reiniciar a especificação para outro recurso, caso exista (#12). Para especificar os relatórios da aplicação (#14), é necessário primeiro salvar a sua especificação (#13).

A aplicação deve ser gerada (#15) assim que todos os padrões da GRN, necessários para especificar a aplicação, forem aplicados. Após a geração da aplicação pela ferramenta de instanciação, o engenheiro de aplicação deve solicitar a criação das tabelas (#16), que é feita no banco de dados relacional MySQL. Após isso, a aplicação está pronta para ser executada (#17), sendo necessário fechar, em seguida, a janela principal da ferramenta de instanciação (#21).

Para apoiar e facilitar o procedimento de especificação da aplicação, a ferramenta GREN-Wizard fornece algumas informações importantes, como: relatórios dos padrões aplicados (#18), uma descrição geral da linguagem de padrões GRN (#19) e a estrutura da linguagem de padrões (#20).

2.4.2.2 Outros Frameworks

O framework IBM SanFrancisco (*SanFrancisco Component Framework*) é uma coleção de componentes, baseados na linguagem Java, que permite aos desenvolvedores construir aplicações de negócio a partir de partes existentes, ao invés de iniciar do nada (Monday et al., 2000). Como é baseado em Java, aplicações podem ser escritas e executadas em diversas plataformas: Windows NT, OS/400, AIX, Solaris e HP-UX. A arquitetura do framework simplifica o processo de integração de diferentes aplicações que uma empresa possui e é composta de três camadas:

- *Foundation Layer and Utilities* (Camada Base e de Utilitários): é a camada principal e fornece serviços fundamentais tais como: criação de objetos, sincronização, persistência e modelo de desenvolvimento da aplicação. Ela suporta segurança, fornece processamento distribuído de transação e cria uma camada *middleware* entre uma aplicação cliente e o servidor.

- *Common Business Objects - CBO* (Objetos de Negócio Comuns): essa camada é construída acima da Camada Base e de Utilitários e juntas, formam a base do framework. A camada CBO consiste de centenas de objetos que são solicitados pelo domínio do negócio (por exemplo, empresa, moeda, parceiro de negócio, unidades de medidas) e de objetos que são utilizados para proporcionar consistência e interoperabilidade entre as aplicações desenvolvidas com o framework e as aplicações legadas. A camada Processos de Negócio Centrais, situada acima da camada CBO, utiliza as classes dessa camada.

- *Core Business Processes - CBP* (Processos de Negócio Centrais): essa é a primeira camada da arquitetura e fornece a estrutura e os comportamentos necessários para que

> os desenvolvedores adaptem os processos de negócios, definidos nessa camada, para serem utilizados em suas aplicações. Por exemplo, SanFrancisco contém o processo de negócio de contabilidade geral (*General Ledger*), que inclui a arquitetura, o projeto e a lógica *default* para construir uma aplicação de contabilidade geral. O desenvolvedor, ao invés de construir sua aplicação a partir do nada, necessita somente estender essa camada para construir uma aplicação de contabilidade geral que pode ser adaptada às suas necessidades. Cada CBP contém somente os processos e objetos específicos do negócio para seu domínio particular.

Como ocorre no GREN, desenvolvedores de aplicação podem utilizar SanFrancisco em qualquer uma das três camadas.

OmniBuilder (Omnisphere, 2002) é um ambiente de desenvolvimento totalmente orientado a objetos e utiliza componentes de software pré-fabricados para gerar aplicações de negócio em diversas tecnologias (por exemplo, Java, C++, arquitetura em três camadas utilizando CORBA/COM/OLE), a partir de um modelo de negócios. Esse ambiente gerencia todo o ciclo de vida da aplicação.

Todos os conceitos do negócio e de projeto são capturados em OmniBuilder, permitindo o reúso desses por meio do uso de gabaritos do negócio e de padrões de projeto (Gamma et al., 1995), respectivamente.

Bäumer et al. (1997) apresentam um sistema, denominado Gebos, que possui frameworks para apoiar o desenvolvimento de aplicações bancárias, mais especificamente caixa, empréstimo e departamentos de investimentos, assim como facilidades de auto-atendimento; e consiste de 2500 classes na linguagem C++.

A arquitetura do sistema Gebos é composta por diversas camadas, que se preocupam com a distinção entre a seção de negócio (por exemplo, caixa e investimento), o domínio do negócio (por exemplo, conta bancária e produtos) e o contexto do local de trabalho (por exemplo, centro de serviço ao cliente, serviço de caixa e de auto-atendimento na agência ou na casa do cliente, por meio da Internet). Diversas aplicações podem ser baseadas em diferentes seções de negócio e diferentes seções de negócio podem ser baseadas no mesmo domínio do negócio. Duas camadas adicionais (Camada do Núcleo Técnico e Camada da Área de Trabalho) oferecem suporte completo ao sistema Gebos.

O framework Qd+, proposto por Ré (2002), foi desenvolvido na linguagem Smalltalk e utiliza o banco de dados MySQL, e é considerado uma extensão do framework GREN. Como o framework GREN, a construção do framework Qd+ foi baseada em uma linguagem de padrões (isto é, a linguagem de padrões LV) e seguiu o processo de construção de framework definido por Braga e Masiero (2002).

Com a instanciação de Qd+ obtêm-se sistemas no domínio de Leilões Virtuais. A principal diferença entre o Qd+ e o GREN é que o primeiro possui interface Web e uma arquitetura em três camadas. Essa arquitetura possui um navegador na máquina cliente,

que corresponde à **camada de apresentação**, um servidor de Web ou servidor HTTP, que corresponde à **camada de aplicação**, e um servidor de banco de dados, que corresponde à **camada de persistência**.

O framework Qd+ utiliza o padrão MVC (*Model-View-Controller*) (Gamma et al., 1995), que separa as funções da aplicação (*Modelo*) das funções da apresentação (*Visão*) e permite a comunicação entre esses dois tipos de funções por meio de um *Controle*. Por isso, os elementos que compõem a arquitetura são agrupados de acordo com o MVC: o *Modelo*, é representado pelas "Classes da Aplicação" e pela classe "Objeto Persistente", que cuida da persistência dos objetos da aplicação em base de dados relacional MySQL; a *Visão*, é representada pelas "Páginas HTML estáticas" e "Páginas HTML dinâmicas"; e o *Controle*, é representado pelo "Servidor Web Smalltalk", pelo "Formulário Web" e pelos "Componentes do Formulário Web".

2.4.3 Resumo dos Frameworks Estudados

Uma síntese das características dos frameworks estudados é apresentada na Tabela 2.4. Algumas características de alguns frameworks não puderam ser observadas uma vez que, por serem proprietários, não fornecem algumas informações.

A resistência quanto à adoção de frameworks está relacionada à dificuldade em entendê-los (Johnson, 1992). À medida que os desenvolvedores obtém mais experiência com o framework pode ser possível sincronizar, desde o início, o projeto da aplicação mais próximo da infra-estrutura do framework (Basili et al., 1998). Essa experiência pode ser facilitada quando a implementação de frameworks é baseada em linguagem de padrões, como ocorre com o GREN e com o Qd+, diferentemente dos demais frameworks discutidos. Nesse caso, o desenvolvedor do software realiza a análise da aplicação baseando-se nos padrões que são utilizados na construção da infra-estrutura do framework.

O GREN foi selecionado para ser utilizado nos estudos de caso de reengenharia utilizando o processo de reengenharia definido nesta tese (apresentado no Capítulo 3) pois: atende ao domínio de Gestão de Recursos de Negócios, que é um subdomínio do domínio de interesse desta tese; sua construção foi baseada em uma linguagem de padrões; permite reúso de projeto e código, bem como de análise (fornecido pela linguagem de padrões de análise em que é baseado); e foi desenvolvido no mesmo Grupo de Pesquisa deste trabalho. Além disso, como é caixa-cinza, tira proveito das vantagens dos outros dois tipos, fornecendo flexibilidade sem expor todo o código fonte (Braga, 2003).

Tabela 2.4: Resumo dos frameworks estudados

Frameworks/ Características	**Sistema Gebos** (Bäumer et al., 1997)	**GREN** (Braga, 2003)	**IBM SanFrancisco** (Monday et al., 2000)	**OmniBuilder** (Omnisphere, 2002)	**Qd+** (Ré, 2002)
Abrangência	Sistemas bancários	Sistemas de Compra, Venda, Manutenção, Cotação, Locação e Pagamento de Recursos	Sistemas de gerenciamento de armazém, pedido, contabilidade geral e domínios de contas a pagar e contas a receber; Sistemas de Negócios em geral	Sistemas de negócios abertos	Sistemas de Leilões Virtuais
Técnicas utilizadas para extensão (Fayad e Johnson, 2000)	Caixa-preta e Caixa-branca	Caixa-cinza	Caixa-cinza	Caixa-preta	Caixa-branca
Arquitetura alvo	Cliente/Servidor	Stand-alone	Cliente/Servidor	Cliente/Servidor	Três Camadas
Linguagem alvo	C++	Smalltalk	Java	Java, C++, HTML, ASP	Smalltalk
SGBD alvo	Dado não fornecido	MySQL	IBM DB2 e Oracle	SQLServer e Oracle	MySQL
Plataforma alvo	Windows	Windows	Diversas, pois utilizou linguagem independente da plataforma na sua construção	Windows NT, Windows 95	Windows
Utiliza padrões de projeto	Sim	Sim	Sim	Sim	Sim
Utiliza linguagem de padrões	Não	Sim	Não	Não	Sim
Linguagem de Desenvolvimento	C++	Smalltalk	Java	C++, HTML, etc.	Smalltalk e HTML
Propósito	Comercial	Acadêmico	Comercial	Comercial	Acadêmico

2.5 Métodos Ágeis

Autores de diversos métodos (ou seja, *Adaptive Software Development*, XP (*eXtreme Programming*) (Beck, 2000), Scrum (K.Schwaber e Beedle, 2002), família *Crystal*, FDD (*Feature-Driven Development*) (Coad, 1999), DSDM (*Dynamic System Development Method*) (Stapleton, 1997), "Modelagem Ágil" (MA) (Ambler e Jeffries, 2002) e "Programação Pragmática" (Hunt e Thomas, 1999)) se reuniram para observar as características que eles tinham em comum a fim de estabelecer melhores práticas de desenvolvimento de software como uma alternativa aos modelos de processo de software "tradicionais".

Um resultado do encontro foi a formação da "Aliança Ágil" e a publicação de seu manifesto (Beck et al., 2001). Esse manifesto é detalhado por meio de quatro valores e doze princípios que sustentam esses valores, podendo ser encontrados na maioria dos métodos ágeis. De modo geral, esses métodos têm por objetivo melhorar a eficiência da organização em desenvolver software enquanto buscam diminuir a "burocracia" do desenvolvimento ((Maurer e Martel, 2002) **apud** (Nakagawa, 2002)).

Os quatro valores definidos no manifesto ágil têm como objetivo incentivar o abandono de práticas que tornam o desenvolvimento oneroso:

(i) **Pessoas e interações** são mais importantes do que processos e ferramentas [7];

(ii) **Software operacional** é mais importante do que documentação abrangente;

(iii) **Participação contínua dos clientes** é mais importante do que negociação;

(vi) **Resposta a mudanças** é mais importante do que seguir um plano.

Em geral, esses valores determinam que confiar em interações entre pessoas facilita o compartilhamento de informação e as mudanças rápidas no processo quando necessárias; usar software operacional permite verificar quão rápido os resultados são produzidos e fornece rápido *feedback*; participação dos clientes significa que todos (patrocinador, cliente, usuário e desenvolvedor) são da mesma equipe e com as diferentes experiências e habilidades de cada um é possível mudar as direções do projeto mais rapidamente, quando necessário, produzindo resultados mais apropriados e projetos mais baratos (Highsmith e Cockburn, 2001).

Resumidamente, os doze princípios consistem de: 1) satisfazer os clientes entregando versões contínuas do software o mais cedo possível; 2) permitir mudanças de requisitos em qualquer fase do desenvolvimento; 3) entregar versões do software, que funcionam adequadamente, em curtos períodos de tempo; 4) proporcionar trabalho em conjunto entre desenvolvedores e clientes; 5) fornecer infra-estrutura necessária para que os indivíduos se tornem motivados e desempenhem o trabalho esperado; 6) proporcionar comunicação face a face; 7) alcançar a medida primária de progresso, que é o software operacional; 8) manter harmonia entre clientes, desenvolvedores e usuários; 9) preocupar-se com a qualidade técnica e com a execução de bons projetos; 10) projetar com simplicidade; 11) permitir que equipes da própria organização participem da definição da arquitetura, requisitos e projeto do sistema; e 12) permitir que as equipes, em intervalos regulares, expressem como podem se tornar mais efetivas.

Entrega do software com atraso e com orçamento superior ao estimado foi a principal motivação para o surgimento dos métodos ágeis. Esses métodos foram desenvolvidos principalmente para produzir software com qualidade e em curto espaço de tempo, a fim

[7]No contexto desta tese, a interação entre clientes e engenheiros de software é imprescindível para o sucesso da reengenharia; no entanto, há também processos e ferramentas que também colaboram para isso.

de atender à demanda relacionada à construção de software rápido, em "tempo de Internet", sendo um novo estilo de desenvolvimento para resolver problemas relacionados a mudanças que ocorrem com freqüência (Cockburn e Highsmith, 2001). Em geral, essas mudanças se referem a necessidades do negócio, do cliente e tecnológicas (Abrahamsson et al., 2003).

Como problemas relacionados a entrega do software com atraso e com custo além do estimado e problemas relacionados a mudanças de requisitos também ocorrem na reengenharia de sistemas, notou-se que práticas ágeis também poderiam ser consideradas no contexto de reengenharia, como apresentado no Capítulo 3 desta tese.

Abrahamsson et al. (2002) definem um processo ágil como sendo: 1) incremental (versões pequenas do software, com ciclos curtos e rápidos de desenvolvimento); 2) cooperativo (clientes e desenvolvedores trabalham constantemente juntos); 3) direto (o método é bem documentado, fácil de aprender e de modificar), e 4) adaptável (capaz de atender mudanças a qualquer momento do desenvolvimento).

Diferentemente de abordagens tradicionais de desenvolvimento, métodos ágeis preocupam-se em gerar versões de software mais cedo usando principalmente técnicas, como: programação em pares, refatoração e clientes presentes no desenvolvimento como membros da equipe (Reifer, 2002). Permitem também, como mencionado, mudanças nos requisitos ao longo do ciclo de desenvolvimento do software (Cohn e Ford, 2003), reduzindo o seu custo (Highsmith e Cockburn, 2001). Atualmente, métodos que não permitem mudanças ao longo do ciclo de desenvolvimento não refletem as condições do negócio, podendo levar a empresa à falência (Highsmith e Cockburn, 2001).

Sucintamente, métodos ágeis: 1) incentivam a elaboração de pouca documentação do sistema, suficiente para entender e para criar o seu projeto (quando há muita documentação existe grande probabilidade dela tornar-se desatualizada), sendo que a maioria da documentação escrita é substituída por comunicação informal entre membros da equipe e entre a equipe e os clientes, enfatizando o conhecimento tácito ao invés do conhecimento explícito ((Cockburn e Highsmith, 2001) **apud** (Chau et al., 2003)); 2) facilitam a incorporação de mudanças nos requisitos; 3) aconselham que a equipe de desenvolvimento seja pequena; 4) preocupam-se com soluções simples; 5) fornecem versões do software em intervalos freqüentes (a cada hora, a cada dia, ou mais usualmente, a cada mês ou a cada bimestre); 6) proporcionam constante interação e cooperação dos usuários; 7) aconselham que desenvolvedores e representantes dos usuários sejam bem informados, competentes e autorizados para tomar decisões e 8) realizam testes no software constantemente (Ambler e Jeffries, 2002; Beck, 2000; Highsmith e Cockburn, 2001). Os proponentes de métodos ágeis afirmam que suas características principais são simplicidade e velocidade (Beck, 1999; Highsmith e Cockburn, 2001), tendo como principal intuito apoiar a produção rápida e antecipada de código operacional (Turk et al., 2002).

De acordo com Turk et al. (2002), os passos de métodos ágeis e os objetivos dos projetos são determinados dinamicamente, baseados na análise: 1) da experiência obtida com passos

do processo executados previamente; 2) das experiências similares obtidas fora do projeto; e 3) das mudanças nos requisitos e no ambiente de desenvolvimento. Turk et al. (2002) afirmam ainda que a agilidade de um processo é determinada pela maneira como uma equipe pode adaptá-lo dinamicamente, baseando-se nas mudanças do ambiente e nas experiências dos desenvolvedores.

Williams e Cockburn (2003) indicam as circunstâncias adequadas para o uso de métodos ágeis em um projeto de desenvolvimento de software: 1) o sistema a ser criado não é de missão crítica; 2) os requisitos mudam com freqüência; 3) a equipe de desenvolvimento é pequena, eficiente e situada na mesma localidade; e 4) os clientes, usuários e especialistas no domínio do problema são acessíveis, quando necessário.

Com o surgimento de diversos métodos ágeis, desenvolvedores não têm conhecimento nem da existência de muitos deles nem da capacidade de serem adequados às diversas situações de desenvolvimento de software. Por isso, Abrahamsson et al. (2003) compararam analiticamente nove métodos ágeis quanto: 1) ciclo de vida do desenvolvimento de software (ou seja, quais estágios do ciclo de vida o método cobre); 2) gerenciamento de projeto (ou seja, se o método apóia ou não atividades de gerenciamento de projeto); 3) princípios abstratos versus orientação concreta (ou seja, se o método fornece ou não como tarefas específicas devem ser executadas); 4) universalmente pré-definida versus apropriada às situações (ou seja, se o método atende a todas as situações de desenvolvimento ágil); e 5) evidência empírica (ou seja, se o método fornece evidência empírica de seu uso).

A partir da análise realizada, Abrahamsson et al. (2003) observaram que a maioria dos métodos cobrem desde a especificação dos requisitos até o teste do sistema e somente dois [8] deles fornecem cobertura completa no ciclo de vida do software (ou seja, desde a concepção do projeto até o sistema em uso). No entanto, nenhum método avaliado foi preciso nesse aspecto. Embora a maioria dos métodos estudados forneçam apoio ao gerenciamento de projeto, ainda falta apoio preciso. A maioria dos métodos não apresenta explicitamente como as tarefas relacionadas às práticas, atividades e artefatos devem ser executadas. Isso mostra que a comunidade ágil está mais preocupada em obter aceitação dos valores propostos do que em oferecer orientação de como usar esses valores. Apenas dois dos nove métodos mencionam como deve ser usado em diferentes situações de desenvolvimento de software ágil e apenas três possuem alguma evidência empírica de uso, sendo necessários mais trabalhos empíricos para observar as implicações de diferentes métodos ágeis em diferentes empresas e situações. Isso motivou o detalhamento das atividades do processo ágil de reengenharia proposto nesta tese (Capítulo 3) e a criação de um pacote de experimentação (Apêndice A) para analisar tal processo.

Os valores e práticas do método XP são comentados a seguir, por ser o método que colaborou efetivamente para a divulgação das abordagens ágeis. Uma vez que XP é o

[8] Ou seja, DSDM e ISD (*Internet-Speed Development*) ((Cusumano e Yoffie, 1999) **apud** (Abrahamsson et al., 2003)).

método ágil mais conhecido e, de acordo com Boehm (2002) as práticas e os valores de XP ajudam a determinar se uma organização está usando um método ágil ou não, suas práticas foram comparadas com as características do processo de reengenharia definido nesta tese, objetivando analisar suas características ágeis. Nenhum processo ágil de reengenharia foi encontrado na literatura até o momento da escrita desta tese.

- **eXtreme Programming**

Beck (2000) afirma que as idéias de XP não são novas e que a maioria delas são tão velhas quanto programar. Por isso, XP é considerado conservador, ou seja, todas as suas técnicas foram provadas durante décadas. XP é uma metodologia para ser utilizada por equipes de desenvolvimento de software de tamanho pequeno e médio. De acordo com Highsmith e Cockburn (2001), o que é novo em metodologias ágeis não são as práticas que elas utilizam mas sim o reconhecimento de pessoas como responsáveis para o sucesso dos projetos. Para permitir isso, equipes de projetos ágeis concentram-se em aumentar tanto os níveis de competência individual quanto os de colaboração [9] entre os seus membros (Cockburn e Highsmith, 2001).

XP é definido como um conjunto de quatro valores (**comunicação** contínua entre gerentes, desenvolvedores e clientes; **simplicidade**, com o uso de soluções minimalistas, durante o desenvolvimento do projeto para não ser feito a mais do que é necessário; ***feedback*** constante sobre o estado do sistema por meio de testes de unidade e funcional, programação em pares e usuários experientes; e **coragem** para tratar de problemas que surgem repentinamente, fazendo ajustes significativos no sistema a fim de garantir que funcione adequadamente) e doze práticas que guiam as atividades da equipe envolvida no projeto. Essas práticas estão intrinsicamente relacionadas e preocupam-se principalmente com o gerenciamento do escopo da iteração, pois XP é baseado no ciclo de vida iterativo e incremental, e com as atividades de implementação e testes. Ressalta-se que as iterações têm duração curta, por exemplo, entre uma e quatro semanas, e são sempre de tamanho fixo.

De acordo com Beck (2000), os quatro valores de XP são muito vagos para ajudar o desenvolvedor decidir quais práticas usar. Para permitir isso, os valores foram refinados em princípios concretos, ou seja: **rápido *feedback*** - interprete o estado atual do sistema e faça os ajustes necessários o mais rápido possível, **assuma simplicidade** - faça um bom trabalho pensando em soluções para resolver os problemas atuais, **mudança incremental** - resolva qualquer problema em diversas mudanças menores, **"abrace" mudanças** - aceite mudanças em qualquer estágio do projeto e **trabalho de qualidade** - faça o trabalho sempre pensando em qualidade.

[9] Cockburn e Highsmith (2001) distinguem colaboração de comunicação: comunicação é o envio e o recebimento de informação, enquanto que colaboração é trabalhar ativamente junto para entregar um produto ou tomar uma decisão.

As doze práticas que norteiam o desenvolvimento em XP são citadas e descritas resumidamente a seguir:

1. *Jogo do planejamento* - determina o escopo da próxima versão do sistema, levando em consideração prioridades do negócio e estimativas do pessoal técnico. Caso não seja possível cumprir tudo o que foi planejado, o cliente deve determinar o que é menos importante para ser desenvolvido na próxima versão.

2. *Versões pequenas* - coloca rapidamente o sistema em funcionamento, liberando novas versões em pequenos intervalos de tempo. Com isso, o cliente pode verificar se o sistema atende os requisitos inicialmente definidos.

3. *Metáfora* - guia todo o desenvolvimento do software por meio de analogias com outros sistemas (não necessariamente um software) conhecidos pela equipe, facilitando o entendimento do projeto e criando um "vocabulário comum" entre os desenvolvedores.

4. *Projeto simples* - o sistema deve ser projetado o mais simples possível para atender as necessidades atuais e não deve haver preocupação do que pode ser usado amanhã. Complexidade extra é removida do projeto assim que é descoberta.

5. *Testes constantes* - o desenvolvimento do software é guiado por testes. Testes de unidade são criados antes de escrever o código e são executados constantemente. O desenvolvimento do software continua somente se não houver erros. Clientes são responsáveis por elaborar testes funcionais de aceitação. Teste em XP é a maior fonte de *feedback* de qualidade do software (Jeffries, 1999). Maiores detalhes sobre essa prática são encontrados na Subseção 2.7.2.

6. *Refatoração constante* - programadores reestruturam o sistema sem alterar o seu comportamento visando remover código duplicado, melhorar a comunicação entre programadores, simplificar o código ou adicionar flexibilidade. Essa prática deve ser considerada sempre que o programador observar que o código está se tornando complexo e confuso.

7. *Programação em pares* - toda produção do código (ou seja, projeto, codificação e teste) é feita com dois programadores em uma máquina para que um ajude o outro, aumentando a produtividade da equipe. Essa prática proporciona que a equipe fique mais integrada e que os seus membros aprendam uns com os outros.

8. *Propriedade coletiva do código* - todos são responsáveis por todo o código do projeto. Dessa forma, qualquer membro da equipe pode mudar qualquer parte do código do sistema, em qualquer momento. Preferencialmente, essa atividade deve ter suporte de uma ferramenta de controle de versão, por exemplo CVS (Concurrent Versions System, 2004).

9. *Integração contínua* - compilar e integrar o código do sistema várias vezes ao dia para que possíveis erros de integração sejam detectados e isolados mais facilmente. Os programadores envolvidos devem corrigir os erros que aparecerem e executar todos os casos de testes já existentes e aqueles criados por eles, até que nenhum erro seja encontrado.

10. *Semana de 40 horas* - não trabalhar mais do que 40 horas na semana para que todos permaneçam descansados e motivados para fazer um bom trabalho.

11. *Cliente presente* - ter sempre na equipe de desenvolvimento a presença de um cliente ou usuário do sistema para responder às questões que podem surgir. Além dessa tarefa, ele escreve as estórias que compõem a funcionalidade do sistema, as prioriza de acordo com o valor de negócio de cada uma e escolhe em qual versão elas serão implementadas, como já mencionado na prática *jogo do planejamento*.

12. *Padrões de codificação* - programadores escrevem todo o código fonte do sistema de acordo com um padrão de codificação, visando facilitar a comunicação entre a equipe por meio do código. Isso facilita as práticas *propriedade coletiva do código* e *refatoração*.

Em Beck (1999), XP é descrito sob os seguintes aspectos: ciclo de desenvolvimento, estórias, versões, iteração, tarefa e teste. O ciclo de desenvolvimento de XP pode variar de anos a dias. O cliente decide o que estará presente na próxima versão do sistema, selecionando as características mais importantes do sistema (chamada de estórias em XP) a partir de todas as estórias possíveis. Isso é feito até obter todo o sistema implementado. O cliente pode selecionar um conjunto de estórias e o programador calcula a data final de entrega da versão do sistema, ou pode selecionar uma data e o programador calcula o orçamento, escolhendo estórias até cumprir o que foi previsto no orçamento.

O objetivo de cada iteração em XP é colocar em produção novas estórias que são implementadas, testadas e que tornam prontas para serem utilizadas pelos clientes. O processo inicia com um plano, que determina as estórias para serem implementadas e as divide em tarefas para que as equipes possam implementá-las. O programador responsável pela equipe primeiro encontra um parceiro, porque toda a produção de código é escrita com duas pessoas em uma máquina. Se existe qualquer dúvida sobre o escopo ou sobre a abordagem de implementação, os parceiros têm uma reunião curta (quinze minutos) com o cliente e/ou com o programador mais experiente sobre a parte do código que será manipulada durante a implementação. Se houver necessidade de simplificar o código e torná-lo mais fácil de entender, a equipe pode refatorá-lo.

O programador responsável pela tarefa estima o tempo que gastará para concluí-la. Se alguma equipe está sobrecarregada e a outra não, a equipe desocupada assume mais tarefas. Depois de finalizar cada tarefa, a equipe deve integrar o código produzido e testá-lo com o sistema atual. Todos os testes devem ser executados e, caso algum teste não execute, o código não pode ser integrado.

Enquanto a equipe está implementando, o cliente está especificando os testes funcionais, baseando-se no que poderia convencê-lo de como as estórias de uma iteração devem se comportar para estar operando corretamente. No final da iteração, todos os testes de unidade e funcionais devem ser executados e a equipe deve estar pronta para a próxima iteração. Os testes de unidade são escritos pelos próprios programadores antes de escreverem o código. Todos os testes de unidade são armazenados permanentemente e devem ser executados automaticamente todas as vezes que uma parte do código produzida for integrada ao sistema.

Isso faz com que as equipes ganhem confiança sobre o comportamento do seu sistema no decorrer do tempo.

2.6 Processos de Software

A importância de processos de software para apoiar a engenharia de software é comentada por diversos autores (Pressman, 2005; Rocha et al., 2001; Sommerville, 2000). Esses processos são associados a métodos, técnicas, ferramentas, normas e modelos de maturidade (por exemplo, ISO/IEC 15504 (ISO, 1998), ISO/IEC 12207 (ISO, 1995), ISO 9001 (ISO, 1994), ISO/IEC 9126-1 (ISO, 1999b), ISO/IEC 14598-1 (ISO, 1999a), *Capability Maturity Model* (CMM) (Paulk, 1993), *Capability Maturity Model Integration* (CMMI) (Carnegie Mellon University, 2002) e MR mps (Modelo de Referência para melhoria de processo de software em empresas brasileiras) (Weber et al., 2004)), que juntos buscam qualidade e produtividade de software no desenvolvimento, na manutenção e na reengenharia. Além disso, incorporam uma estratégia, conhecida também como modelo de processo ou paradigma de engenharia de software ou ainda ciclo de vida (Pressman, 2005), que determina a maneira que os recursos (métodos, técnicas e ferramentas) devem ser aplicados. Nesta tese são definidos dois processos de software, um no contexto de reengenharia e outro no contexto de evolução de frameworks de aplicação, apresentados respectivamente nos Capítulos 3 e 4, que são associados ao arcabouço de reengenharia ágil definido no Capítulo 3.

Diversos modelos de processos de software são encontrados na literatura, por exemplo, seqüencial linear ou em cascata, incremental, espiral, de prototipação e baseado em componentes. Pressman (2005) classifica os modelos incremental e espiral como evolutivos (ou evolucionários). Nesta tese será dado enfoque no modelo de processo incremental - conhecido também como iterativo (Larman, 2004a), pois é o utilizado nos processos definidos nesta tese.

No modelo incremental, o desenvolvimento é organizado em uma série de mini-projetos de tamanho pré-fixado e curto, chamados iterações. A saída de cada iteração é um sistema parcial, testado e executável. Cada iteração inclui atividades de análise, projeto, codificação e teste, isto é, cada iteração preocupa-se com a escolha de um pequeno subconjunto de requisitos e em seguida, projetar, implementar e testar esse subconjunto. Desenvolvimento iterativo é uma prática-chave utilizada pela maioria dos métodos modernos, pois permite mudanças e adaptações em qualquer momento da aplicação do processo. Além disso, qualquer método iterativo pode ser aplicado de uma maneira ágil uma vez que passos pequenos, rápido *feedback* e adaptação são as idéias centrais de desenvolvimento iterativo (Larman, 2004a).

Larman (2004a) apresenta alguns benefícios de desenvolvimento iterativo que podem ser alcançados também no contexto de reengenharia e de evolução de frameworks: 1) reduz falhas

no projeto, melhora a produtividade e garante menores taxas de defeito; 2) reduz previamente riscos no projeto relacionados aos requisitos, aos objetivos, à usabilidade, entre outros; 3) permite o progresso visível do projeto antecipadamente; 4) permite *feedback*, afinidade com o usuário e adaptação nos requisitos, levando a um sistema refinado que se aproxima cada vez mais das necessidades dos interessados; 5) gerencia complexidade, sendo que a equipe não é desmotivada por passos complexos e muito longos; e 6) permite que a aprendizagem obtida em uma iteração seja utilizada para melhorar o próprio processo de software.

De acordo com Fuggeta (2000), a qualidade dos produtos de software está intrinsicamente relacionada à qualidade do processo utilizado. Ressalta-se que tanto a qualidade do produto quanto a qualidade do processo são importantes para garantir que o software atenda às necessidades de seus usuários e para que os prazos e custo inicialmente previstos sejam cumpridos. De acordo com Rocha et al. (2001) a qualidade do software depende das pessoas, da organização e dos procedimentos usados em seu desenvolvimento. A definição e o uso de processos de software envolvem fatores organizacionais, culturais, tecnológicos e econômicos (Fuggeta, 2000).

Do ponto de vista de processos de manutenção de software, diversos têm sido propostos na literatura (Arthur, 1988; Pfleeger e Bohner, 1999; Polo et al., 1999; Yau e Collofello, 1980). Alguns deles (Pfleeger e Bohner, 1999; Yau e Collofello, 1980) consideram análise de impacto como medida válida e necessária para apoiar a manutenção de software (Black, 2001; Zhao et al., 2002), pois ela mostra os efeitos que poderão ser causados pela manutenção no código fonte. Embora existam diversas particularidades em cada processo de manutenção de software, todos eles possuem três passos básicos de manutenção de software, definidos por Boehm (1976): entendimento, modificação e revalidação do software. Esses passos também são observados no processo de evolução de frameworks de aplicação definido nesta tese (Capítulo 4).

Processo de reengenharia é outro tipo de processo de software de interesse deste trabalho e foi discutido na Seção 2.2 deste Capítulo.

2.6.1 Documentação de Processos de Software

Como software, processos de software devem ser documentados de uma maneira sistemática para possibilitar o seu entendimento, bem como facilitar o seu uso e a sua evolução. Existem diversos formatos (ou padrões ou modelagens) de documentação de processos na literatura (ISO, 1995; Kruchten, 2000; OMG, 2005), ((Christie, 1993; Franch e Ribó, 1999; Jaccheri et al., 1998) **apud** (Genvigir et al., 2003)). No entanto, não existe um consenso sobre a taxonomia dos elementos que compõem os processos e nem sobre como esses elementos podem ser modelados (Genvigir et al., 2003). Dois formatos de documentação existentes são discutidos nesta subseção: SPEM (*Software Process Engineering Metamodel*) (OMG, 2005) e RUP (*Rational Unified Process*) (Kruchten, 2000).

Padrões de documentação de processos de software definem o processo utilizado para produzir artefatos de software. Isso significa definir os procedimentos e as ferramentas envolvidas na elaboração dos artefatos, bem como os procedimentos de verificação e refinamento, que garantem a produção de artefatos de boa qualidade (Sommerville, 2000). Os principais objetivos para documentar um processo de software são: possibilitar a comunicação e o entendimento efetivo do processo, facilitar o reúso do processo (padronização), apoiar a evolução do processo e facilitar o gerenciamento do processo ((Humphrey e Kellner, 1989; Kellner e Hansen, 1988) **apud** (Genvigir et al., 2003)). Salienta-se que um processo de software bem documentado e bem definido colabora para o sucesso dos projetos (Kruchten, 2000). Por isso, nesta tese, procurou-se utilizar um formato de documentação de processos, no caso alguns elementos fundamentais que compõem o arcabouço do RUP, que pudesse melhor representar a documentação dos processos propostos, de forma simples e organizada.

Software Process Engineering Metamodel (SPEM), como o próprio nome diz, é um metamodelo definido pela OMG (*Object Management Group*) para descrever um processo de software concreto ou uma família de processos de desenvolvimento de software relacionados. Contém a definição de um conjunto mínimo de elementos de modelagem de processos, necessários para qualquer processo no contexto de desenvolvimento de software. A especificação de SPEM é estruturada como um perfil [10] UML (*Unified Modeling Language*) (Fowler e Scott, 1997; OMG, 2003) e, como a própria UML, SPEM também fornece um metamodelo completo baseado no meta-metamodelo MOF (*Meta-Object Facility*), que é a tecnologia adotada pela OMG para definir meta-dados. MOF facilita a troca de informação com ferramentas UML e repositórios/ferramentas baseadas nele.

Na Figura 2.7 é apresentada uma arquitetura de modelagem em quatro camadas, como definido pela OMG. Um processo que está em uso no mundo real é apresentado no nível M0. A definição do processo correspondente está no nível M1. Ressalta-se que tanto um processo genérico quanto o RUP e a instanciação desse processo em um determinado projeto estão no nível M1. Como SPEM é um metamodelo, está no nível M2 e é um gabarito para o nível M1.

Embora existam diversos elementos da UML que foram estendidos pelo SPEM (pacote `SPEM_Foundation`) e outros que foram criados para formar a estrutura e a semântica necessárias para a engenharia de processos de software (pacote `SPEM_Extensions`), nesta subseção serão comentados somente os elementos do modelo conceitual e os da estrutura do processo. Além desses, há também os elementos básicos (*ExternalDescription*, *Guidance*), as dependências, os elementos do processo (*Package*, *ProcessComponent*, *Process* e *Discipline*) e os do ciclo de vida do processo (*Phase*, *Lifecycle*, *Iteration* e *Precondition and Goal*).

[10] É um tipo de variante da UML que usa mecanismos de extensão (por exemplo, estereótipos) de uma maneira padronizada para um determinado propósito (por exemplo, análise e projeto) ou fornecem mecanismos que especializam a UML para um alvo específico (tal como Java, C++ e Corba), visando incorporar no modelo semântica específica do domínio, não adicionando novos conceitos fundamentais.

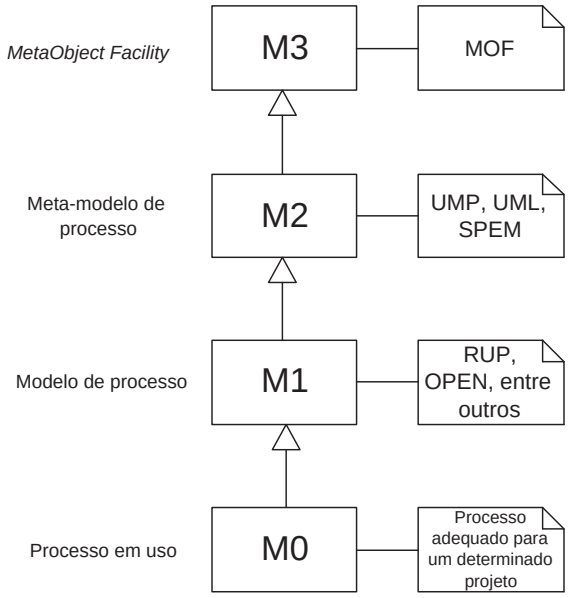


Figura 2.7: Níveis de modelagem (adaptado de (OMG, 2005))

Com relação ao modelo conceitual, a idéia central do SPEM é que um processo de desenvolvimento de software é uma colaboração entre entidades abstratas ativas chamadas papéis [11] de processo (*process roles*) que executam operações, chamadas atividades, em entidades concretas, chamadas produtos de trabalho (*work products*) ((Jacobson e Jacobson, 1995) **apud** (OMG, 2005)). Como no RUP, atividades utilizam artefatos de entrada para serem executadas e produzem artefatos de saída.

Com relação a estrutura do processo, as classes que compõem o pacote da estrutura do processo estão ilustradas na Figura 2.8 [12]. Como mencionado, um papel de processo auxilia na execução de uma ou mais atividades do processo. Além disso, é responsável pela construção de vários produtos de trabalho elaborados durante a aplicação do processo. Cada produto de trabalho possui um tipo (*WorkProductKind*), que descreve a categoria a que pertence (por exemplo, documento de texto, modelo UML e executável). Uma atividade é executada por meio de um ou mais passos (*Step*). A maioria das classes do pacote herdam, de alguma maneira, das classes do pacote central, denominado *Core*. Existem algumas classes intermediárias que fornecem mais expressividade ao diagrama de classes da estrutura do processo, como *WorkDefinition* (é um tipo de operação que descreve o trabalho realizado no processo) [13], *ProcessPerformer* (define um executor para um conjunto de *WorkDefinitions* em um processo), *ActivityParameter* (especifica "como" - entrada ou saída - os produtos de trabalhos são utilizados por uma atividade).

[11] Um papel é uma função que um indivíduo desempenha no processo.

[12] A descrição das cardinalidades das classes desse pacote, feita nesta tese, é baseada no texto disponível em (OMG, 2005) e não nas cardinalidades apresentadas na Figura 2.8.

[13] Além de atividade, algumas classes do pacote do ciclo de vida do processo como, fase, iteração e ciclo de vida, também são suas subclasses, mas não estão apresentadas na Figura 2.8.

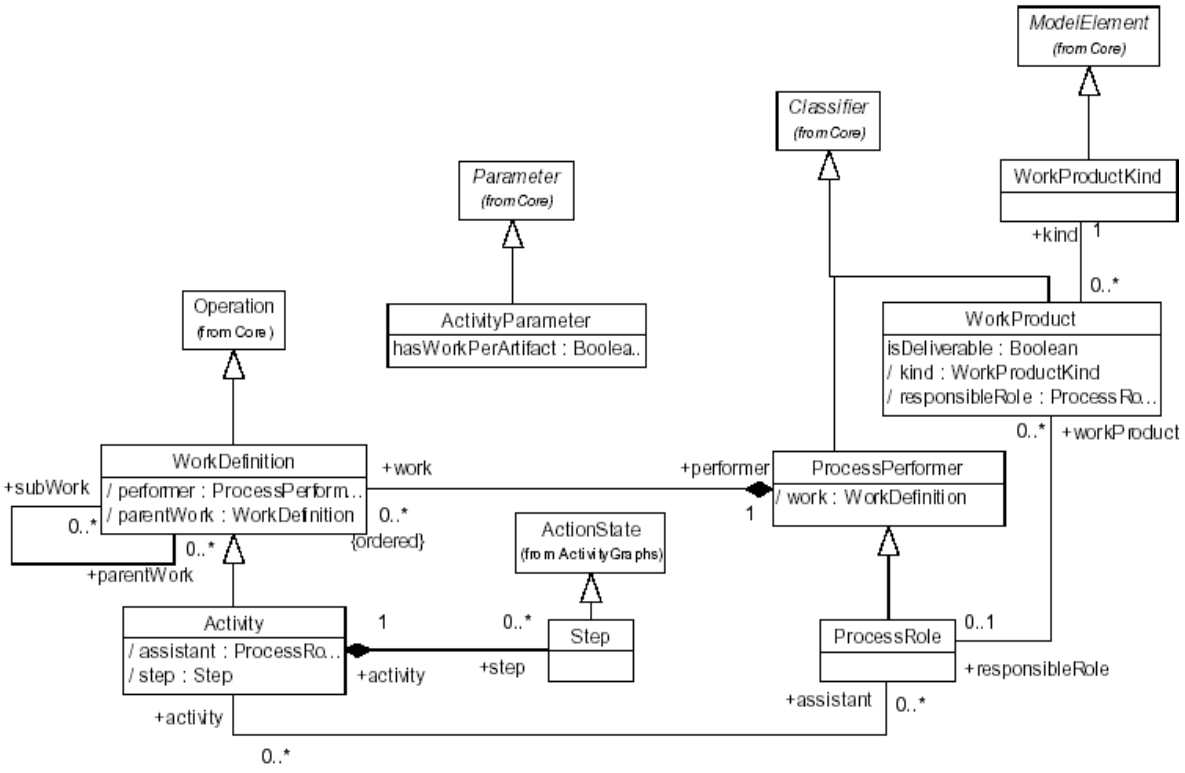


Figura 2.8: Pacote da estrutura do processo (OMG, 2005)

Por sua vez, o RUP é um processo de software que fornece uma abordagem disciplinada para determinar tarefas e responsabilidades dentro de uma empresa de desenvolvimento. Além disso, é um arcabouço de processo que pode ser adaptado e estendido para se adequar às necessidades de uma empresa. Nesta tese, será dada ênfase apenas nos elementos fundamentais que compõem o arcabouço do RUP e não em detalhes do comportamento do processo em si, pois são os de interesse para o entendimento da documentação dos processos propostos neste trabalho. RUP possui uma estrutura com duas dimensões, de acordo com a Figura 2.9.

- Eixo horizontal: representa o tempo e mostra aspectos do ciclo de vida do processo;
- Eixo vertical: representa o aspecto estático do processo, ou seja, sua descrição em termos de componentes de processo, atividades, disciplinas (que agrupam logicamente as atividades do processo), artefatos e papéis (*roles*).

A primeira dimensão, eixo horizontal, representa o aspecto dinâmico do processo e é expressa em termos de ciclos, fases (Concepção - *Inception*, Elaboração - *Elaboration*, Construção - *Construction* e Transição - *Transition*), iterações e marcos de referência (*milestones*). Cada fase é concluída por um marco de referência que indica a situação do projeto e se o engenheiro de software deve proceder, encerrar ou alterar o percurso do processo.

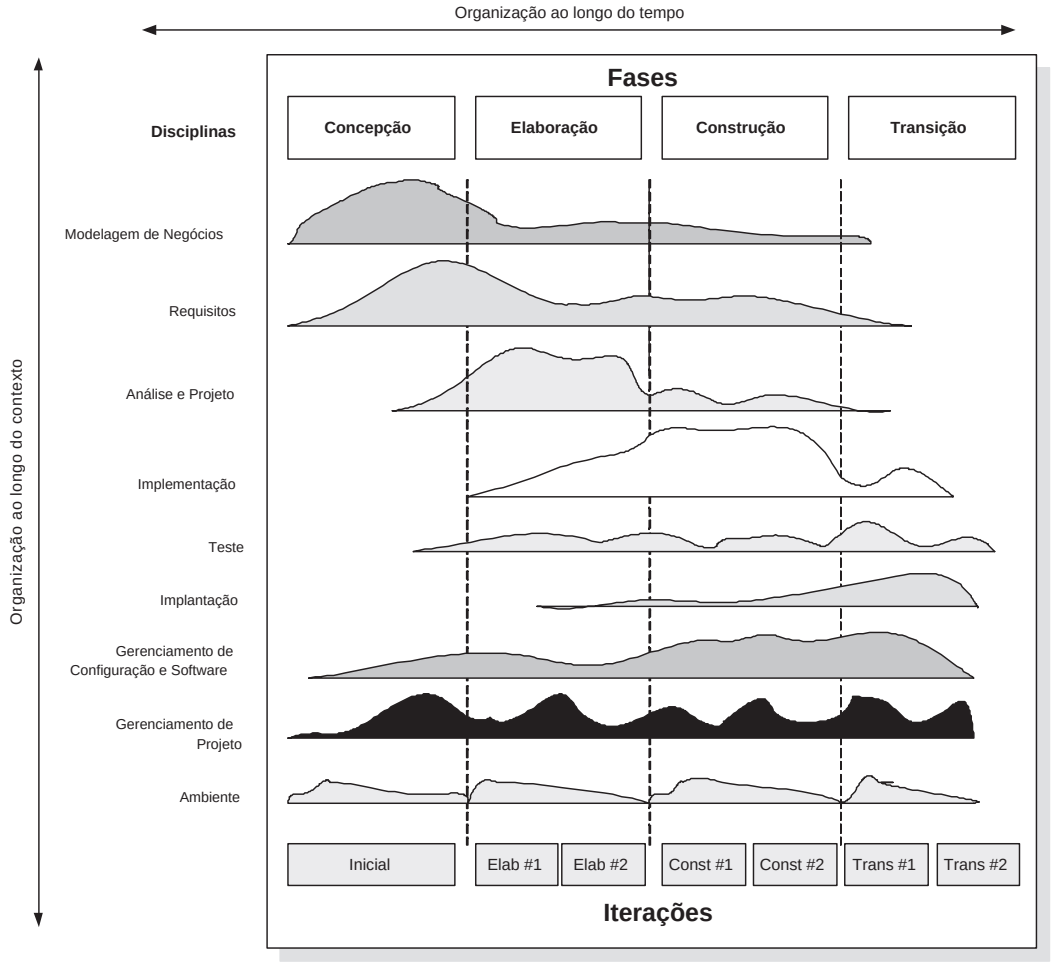


Figura 2.9: Estrutura do RUP (adaptado de Kruchten (2000))

As fases constituem um ciclo de desenvolvimento e produzem uma versão do software. Um produto é criado em um ciclo de desenvolvimento inicial e em seguida evoluirá nas próximas versões pela repetição da seqüência das fases de Concepção (C), Elaboração (E), Construção (C) e Transição (T), como ilustrado na Figura 2.10.

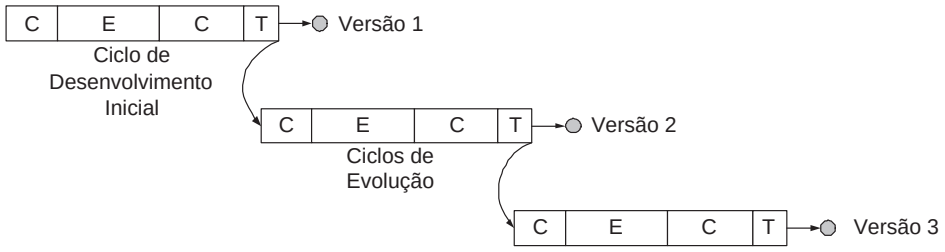


Figura 2.10: Ciclos de evolução (Kruchten, 2000)

Um processo descreve "quem" está fazendo "o quê", "como" e "quando". Para documentar isso, o RUP utiliza quatro elementos primários de modelagem:

- Papel (representa "quem"): define o comportamento e responsabilidades de um indivíduo ou grupo de indivíduos trabalhando juntos em uma equipe. O comportamento é definido por meio de atividades que o indivíduo executa. Um indivíduo pode desempenhar mais do que uma função, ou seja, pode agir com diversos papéis e um papel pode ser desempenhado por diversos indivíduos. Exemplos de papéis: analista de sistemas, projetista, programador, testador, entre outros.

- Atividades (representa "como"): uma atividade é uma unidade de trabalho que um indivíduo pode executar e que produz um resultado significativo no contexto do projeto. A atividade tem um propósito claro, geralmente expresso em termos de criar ou atualizar artefatos, tais como um modelo, uma classe ou um plano. Atividades são divididas em passos para facilitar a sua execução.

- Artefatos (representa "o quê"): os artefatos são classificados em artefatos de entrada e de saída. Um artefato é uma informação que é produzida, modificada ou usada por um processo.

- Disciplina (representa "quando"): somente papéis, atividades e artefatos não constituem completamente um processo. É necessário uma maneira de descrever significativamente as seqüências das atividades que produzem algum resultado de valor e mostrar as interações entre os papéis. Uma disciplina é uma seqüência de atividades que produz um resultado significativo.

Existem outros elementos que são adicionados às atividades e aos artefatos para tornar o processo mais fácil de entender e usar:

- planos de iteração (*iteration plans*): documenta as atividades que serão efetivamente executadas durante a iteração;

- diretrizes (*guidelines*): são regras, recomendações ou heurísticas que são ligadas às atividades, passos ou artefatos e têm como objetivo apoiá-los; são utilizadas também para validar a qualidade dos artefatos na forma de *checklists* associadas aos artefatos ou para revisar atividades;

- gabaritos (*templates*): são protótipos de artefatos para apoiar a sua criação;

- guias da ferramenta de apoio computacional (*tool mentors*): são um meio adicional de guiar o engenheiro de software, mostrando como usar uma ferramenta de software específica; e

- conceitos (*concepts*): são conceitos importantes, introduzidos em seções separadas do processo e podem ser definidos tanto para o processo quanto para o produto.

Como há um grande número de modelos de processos, formatos e padrões e cada um utiliza terminologia diferente para representar o mesmo elemento do processo, em (OMG, 2005) há um mapeamento das terminologias de diferentes fontes com relação às terminologias utilizadas pelo SPEM. Na Tabela 2.5 ilustra-se um fragmento desse mapeamento, apresentando a correspondência entre os elementos do SPEM e do RUP. Como pode ser visto, diversos elementos do SPEM podem ser mapeados para elementos do RUP. Assim, a documentação dos processos propostos nesta tese, descrita por meio de alguns elementos fundamentais do RUP, também pode ser representada em SPEM.

Tabela 2.5: Fragmento da tabela de mapeamento de terminologias (OMG, 2005)

SPEM	Papel de processo	Atividade Passo	Produto de Trabalho	Disciplina	Ciclo de Vida	Fase	Iteração	Orientação (*Guidance*)
RUP	Papel	Atividade Passo	Artefato	Disciplina	Processo	Fase	Iteração	Diretrizes, Guia de Ferramentas, Gabaritos
...	...	...	...	...	...	...	...	...

2.7 Garantia da Qualidade de Software

A garantia da qualidade do software é a principal atividade com a qual uma equipe de desenvolvimento, manutenção e reengenharia deve se preocupar para entregar um produto confiável a seus usuários. Essa atividade deve ser conduzida em todas as fases do processo de software para que os defeitos introduzidos sejam eliminados na fase em que surgiram, a fim de evitar o aumento dos custos da remoção desses em fases posteriores da sua criação. Dentre as atividades de garantia de qualidade de software estão as de VV&T (Verificação, Validação e Teste), as quais têm como objetivo minimizar a ocorrência de erros e riscos associados (Maldonado e Fabbri, 2001a). Isso é considerado no processo ágil de reengenharia proposto nesta tese (Capítulo 3), sendo que atividades VV&T são utilizadas desde o início da reengenharia e apóiam o entendimento do sistema legado e a identificação de regras do negócio embutidas no código fonte, bem como a validação do sistema alvo.

Verificação tem como objetivo garantir que o software está sendo desenvolvido corretamente. Por outro lado, validação tem como objetivo garantir que o software que está sendo desenvolvido é o software correto, de acordo com os requisitos especificados pelos usuários (Sommerville, 2000). O teste de software consiste na execução do produto de software visando verificar a presença de erros [14] e aumentar a confiança de que tal produto esteja

[14]Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro ((IEEE, 1990) **apud** (Maldonado e Fabbri, 2001a)).

correto (Maldonado e Fabbri, 2001b). A atividade de teste é importante para a garantia da qualidade tanto do produto quanto do processo.

Diversos autores (Harrold, 2000; Pressman, 2005; Rocha et al., 2001; Sommerville, 2000) comentam a importância de atividades de teste, no entanto, ressaltam que nem sempre são praticadas devido ao tempo e custo associados. Testar um sistema completamente, em geral, é impossível; o objetivo é tentar criar testes tão completos quanto possível. Um bom caso de teste é aquele que tem alta probabilidade de encontrar um erro ainda não descoberto e um teste bem sucedido é aquele que descobre um erro (Myers, 2004a). Para apoiar a criação de testes mais efetivos, critérios de teste são utilizados. Mais especificamente, um critério de teste serve para selecionar e avaliar casos de teste de forma a aumentar as possibilidades de revelar a presença de erros ou, quando isso não ocorre, estabelecer um nível elevado de confiança na correção do produto (Maldonado e Fabbri, 2001a).

Os testes podem ser aplicados em três fases de teste distintas, de acordo com o escopo: de unidade, de integração e de sistema (Pressman, 2005). Os testes de unidade preocupam-se em testar cada unidade do programa para garantir que a implementação de cada uma esteja correta. Mas, testar cada unidade não garante que elas funcionem corretamente quando trabalham juntas. Assim, é realizado o teste de integração, que garante que as interfaces entre as unidades do programa funcionem sem erros. Por fim, é realizado o teste de sistema, que garante que a aplicação propriamente dita e os demais elementos que a compõem (sistema operacional, banco de dados, entre outros) se comuniquem adequadamente entre si e que os requisitos não funcionais estabelecidos (desempenho, confiabilidade, entre outros) sejam atendidos pela aplicação.

Existem diversos tipos de técnicas de teste que são fundamentadas de acordo com a origem das informações utilizadas para estabelecer os requisitos de teste, por exemplo, técnica funcional, estrutural e baseada em erros. Será dado enfoque apenas na técnica de teste funcional, por ser a de interesse desta tese e utilizada pela abordagem de reúso de teste proposta (Capítulo 5) e utilizada no arcabouço de reengenharia ágil definido (Capítulo 3).

A técnica funcional ou caixa preta (*black-box*) aborda o software de um ponto de vista macroscópico e estabelece os requisitos de teste com base na especificação (isto é, consideram-se apenas as entradas, saídas e o estado do programa), não preocupando-se com o comportamento e com a estrutura interna do software (Myers, 2004a; Pressman, 2005).

Sabe-se que cada critério de teste contribui com um conjunto de casos de teste particular. Assim sendo, é necessário estabelecer estratégias de teste que possuam critérios funcionais e estruturais para que se possa obter um conjunto de teste capaz de revelar um número maior de erros, uma vez que, de acordo com Myers (2004a) e Roper (1994), critérios de teste funcionais e critérios de teste estruturais devem ser utilizados em conjunto para que um complemente o outro. No entanto, nesta tese serão utilizados apenas critérios de teste funcional, mais especificamente, o Particionamento de Equivalência e a Análise do Valor Limite, que estão discutidos na próxima subseção.

2.7.1 Critérios de Teste Funcional: Particionamento de Equivalência e Análise do Valor Limite

O critério Particionamento de Equivalência divide o domínio de entrada de um programa em um número finito de classes de equivalência (válidas e inválidas), derivando os casos de teste a partir dessas classes. Esse critério visa minimizar o número de casos de teste, selecionando apenas um caso de teste de cada classe de equivalência, pois, em princípio, todos os elementos de uma classe devem se comportar de maneira equivalente (Maldonado e Fabbri, 2001b). Isto é, se um caso de teste de uma determinada classe de equivalência revela um erro, qualquer outro caso de teste nessa classe deveria revelar o mesmo erro (Myers, 2004a).

O critério de Análise do Valor Limite complementa o critério Particionamento de Equivalência e preocupa-se em criar casos de teste que considerem valores diretamente acima e abaixo dos limites das classes de equivalência. De acordo com Myers (2004a), casos de teste que exploram condições de limite são mais eficientes do que aqueles que não exploram.

A vantagem do critério Particionamento de Equivalência é que ele ajuda reduzir o tamanho aparente do domínio de entrada e preocupa-se em criar casos de teste baseados somente na especificação. O método é particularmente bem adequado para aplicações em que as variáveis de entrada são facilmente identificadas, como é o caso de sistemas de informação. Esse critério é de difícil aplicação quando o domínio de entrada do software é simples e o processamento é complexo. Outra desvantagem é que a técnica fornece pouco auxílio para combinar dados de teste (Roper, 1994).

As vantagens e desvantagens do critério Análise do Valor Limite são similares àquelas do critério Particionamento de Equivalência. Uma vantagem adicional é que esse critério fornece mais orientação na criação de dados de teste e esses dados de teste tendem a se manifestar onde falhas podem ser encontradas (Roper, 1994).

Há também uma técnica de teste encontrada na literatura, denominada "Teste de Validação de Entrada" (*Input Validation Analysis*) (Hayes e Offutt, 1999), que identifica os dados de teste que tentam mostrar a presença ou a ausência de falhas específicas, pertinentes a "tolerância a entradas" (ou seja, verifica a habilidade do sistema processar adequadamente valores de entrada esperados e inesperados). Essa técnica não requer projeto ou código do sistema, mas baseia-se apenas na sua especificação, que deve ser escrita em um formato específico.

2.7.2 Teste de Software no Contexto de Métodos Ágeis

Como comentado na Seção 2.5, o propósito de métodos ágeis, como é o caso de XP, é criar sistemas com qualidade em um curto espaço de tempo. Para isso, utilizam sistematicamente teste de unidade e de aceitação.

Em geral, testes de unidade devem ser executados a cada mudança incremental no código para garantir que o sistema continue atendendo o que foi especificado pelo cliente (Myers, 2004b).

Teste é tão importante em XP que testes de unidade são criados antes da escrita do código. Essa prática é considerada pelo arcabouço de reengenharia ágil, definido nesta tese no Capítulo 3, com o apoio da abordagem de reúso de teste proposta no Capítulo 5. Essa maneira de realizar a atividade de teste é chamada de e**X**treme **T**esting (XT) por Myers (2004b) e de *Test-Driven Development* (TDD) por Beck (2002). Esse último é conhecido também como *Test-First Development* (Larman, 2004b). A criação de testes antes da escrita do código obriga o entendimento da especificação e a solução de problemas de ambigüidades antes do início da codificação. Adicionalmente, de acordo com Beck (2002), essa maneira de realizar teste fornece confiança de que o código satisfaz a especificação; permite que o programador conheça o resultado final do código antes do início da codificação; e permite que projetos simples sejam implementados inicialmente e que o código seja refatorado posteriormente, garantindo que o sistema continue atendendo à especificação.

Experiências mostram que teste antes da codificação reduz o custo de corrigir erros no desenvolvimento de software ((McConnell, 1993) **apud** (Ambler, 2002)). Além disso, a maioria dos problemas é incorporada no software durante atividades de comunicação, como elicitação de requisitos e análise, e não em atividades técnicas, como projeto e codificação ((Ambler, 1998) **apud** (Ambler, 2002)). Por isso, é necessário: 1) testar antecipadamente e com freqüência; 2) validar o que foi produzido; 3) obter constante *feedback* relacionado à qualidade do produto (Ambler, 2002).

XT e teste tradicional têm o mesmo objetivo, ou seja, identificar erros no sistema. No entanto, o que os diferencia é o momento em que os testes são criados.

Teste de unidade extremo (*extreme unit testing*) é a abordagem de teste primária em XT e possui duas regras simples: todas as partes do sistema devem possuir testes de unidade antes de sua codificação e todas as partes codificadas do sistema devem passar pelos testes de unidade, criados previamente, antes da versão do sistema ser liberada, isso para ter certeza de que nenhuma parte do código foi danificada.

Teste de aceitação é o segundo tipo de teste de XT e tem a mesma importância que o primeiro (teste de unidade extremo). O propósito desse teste é determinar se o sistema satisfaz outros requisitos como funcionalidade e usabilidade. Diferentemente do teste de unidade extremo, os clientes é que criam os testes de aceitação a partir das estórias e não os programadores. Esse tipo de teste também é considerado pelo arcabouço de reengenharia ágil definido nesta tese (Capítulo 3).

Se os clientes descobrirem diversos erros no sistema eles devem priorizá-los antes de passar a lista de erros para a equipe de desenvolvimento. Depois que os programadores corrigem os erros, os clientes re-executam os testes de aceitação. Com isso, testes de aceitação tornam-se

também teste de regressão (Myers, 2004b). De acordo com Crispin et al. (2001), teste de aceitação possui alguns benefícios:

- garantem que o sistema está funcionando como o cliente deseja;
- dão segurança aos clientes, gerentes e desenvolvedores de que todo o produto está sendo desenvolvido corretamente;
- controlam cada iteração em XP, comprovando que o valor do negócio está presente no sistema;
- podem considerar testes de desempenho e de carga para evidenciar que a estabilidade do sistema satisfaz os requisitos do cliente; e
- não são escritos baseando-se em documentos formais, mas sim na opinião dos clientes que expressam o que eles esperam do sistema.

Jeffries (1999) confirma a importância de XT para manter o sistema funcionando corretamente depois de ser submetido a uma grande mudança relacionada a detalhes do negócio. Os testes de unidade permitem evoluir o sistema rapidamente e com segurança e os testes de aceitação fornecem confiabilidade de que o software permanece funcionando corretamente.

Com relação ao TDD (*Test-Driven Development*), ele motiva o desenvolvimento automatizando os testes antes da escrita do código. TDD possui basicamente duas regras básicas: escreva o código somente se um teste automatizado falhou e elimine código duplicado para melhorar a qualidade do sistema (Beck, 2002).

Larman (2004b) cita algumas vantagens de TDD:

- teste de unidade é realmente escrito: a escrita de teste de unidade não é praticada, em geral, se deixada como uma atividade posterior;
- satisfação de programadores levam à escrita de testes mais consistentes: se o teste é escrito primeiro, o programador tenta escrever o código para ser executado corretamente com o teste escrito previamente e, se isso ocorre, o programador sente-se realizado;
- esclarecimento da interface e do comportamento detalhado do código: isso é obtido durante a escrita do teste e o programador já imagina como a classe ou o método serão implementados;
- verificação provável, repetível e automatizada: ter centenas ou milhares de testes de unidade construídos em alguma ferramenta de teste, durante semanas, fornece verificação significativa da corretude do sistema.

- confiança para mudar as coisas: se alguma mudança foi realizada no sistema, o conjunto de teste de unidade das classes modificadas devem ser executados para saber se a mudança causou ou não algum erro.

De acordo com Myers (2004b), é importante o uso de ferramentas para apoiar a atividade de teste, já que à medida que a aplicação cresce pode ser necessário criar centenas ou até milhares de testes de unidade. Os casos de teste devem ficar disponíveis em um repositório comum para que todos os programadores possam executá-los e/ou atualizá-los após cada mudança incremental no código. Diversas ferramentas que automatizam testes de unidade e de aceitação foram encontradas na literatura. JUnit [15], que faz parte da família de framework de teste XUnit [16], Pounder [17], WebART [18] (Crispin et al., 2001) e J-FuT (Rocha et al., 2004) apóiam teste de unidade, enquanto FitNesse [19], WEBTEST [20], Fit [21], Marathon [22] e AVIGON [23] apóiam teste de aceitação. Nesta tese utilizou-se a ferramenta S-Unit [24], que é um framework de teste para Smalltalk, em um dos estudos de caso de reengenharia apresentado no Capítulo 3, conduzido para avaliar o processo de reengenharia definido nesta tese.

2.8 Considerações Finais

Este capítulo apresentou os conceitos básicos necessários para o entendimento e valorização desta tese. O estado da arte e o da prática dos diferentes temas abordados nesta tese foram discutidos.

Como comentado neste capítulo, notou-se que a preocupação dos trabalhos que consideram padrões na reengenharia está relacionada com o entendimento e com a recuperação da estrutura e da arquitetura do sistema legado. Dessa forma, o entendimento da funcionalidade torna-se dependente da experiência do engenheiro de software no domínio do sistema legado. A partir disso, notou-se a possibilidade de utilizar nesta tese linguagem de padrões de análise na reengenharia de sistemas legados para alcançar reúso não somente das soluções de análise como também do conhecimento do domínio.

Um outro ponto notado é o aumento da produtividade na indústria de software com o uso de frameworks, além da possibilidade de desenvolver sistemas utilizando novas tecnologias. A partir dessas vantagens de frameworks e considerando o interesse de migrar sistemas legados

[15] http://junit.sourceforge.net/
[16] http://xunit.sourceforge.net/
[17] http://sourceforge.net/projects/pounder
[18] http://www.tensegrent.com
[19] http://fitnesse.org/
[20] http://webtest.canoo.com/webtest/
[21] http://fit.c2.com/
[22] http://sourceforge.net/projects/marathonman/
[23] http://www.nolacom.com/avignon/
[24] http://sunit.sourceforge.net/

para novas tecnologias, bem como a ausência de iniciativas de processos e abordagens de reengenharia com ferramentas computacionais de apoio associadas, notou-se a possibilidade de utilizar nesta tese frameworks como apoio computacional, a fim de obter reúso de projeto e de implementação.

Observou-se também ausência de iniciativas de testes associados a padrões de software. Como mencionado neste capítulo, isso é uma preocupação que deve ser considerada, já que padrões são utilizados na Engenharia de Software e atividades de teste fazem parte da garantia de qualidade de qualquer produto desenvolvido. Apesar de haver preocupação de pesquisadores sobre a validação do sistema resultante da reengenharia, poucos métodos de reengenharia abordam isso, porque custo e esforço consideráveis são despendidos em atividades de teste. Esse contexto colaborou para a motivação de parte deste trabalho que trata do reúso de teste. Tal reúso foi observado com a possibilidade de associar recursos de teste aos padrões de linguagens de padrões de análise, mais especificamente, a GRN. Isso colaborará para a "reengenharia guiada por teste".

Dessa forma, o trabalho abordado nesta tese trata de reúso na reengenharia de software, em diversos níveis de abstração, ou seja, análise, projeto, implementação e teste.

Outros problemas enfrentados na reengenharia são a entrega do software com atraso e com custo além do estimado e as mudanças freqüentes de requisitos, como ocorre no desenvolvimento de software. Assim, é necessário ter um arcabouço de reengenharia que amenize isso, por exemplo, que utilize características de métodos ágeis no contexto de reengenharia. Outra carência identificada foi a ausência de processos de evolução específicos para frameworks de aplicação.

A documentação dos processos propostos nesta tese é baseada nos elementos fundamentais que compõem o arcabouço do RUP. Tanto o framework GREN como a linguagem de padrões GRN foram selecionados para serem utilizados pelo arcabouço de reengenharia proposto nesta tese.

O próximo capítulo apresenta um processo ágil de reengenharia baseado em framework e alguns estudos de caso de reengenharia utilizando o framework GREN como apoio computacional. Os dois capítulos subseqüentes apresentam um processo de evolução de frameworks de aplicação e uma abordagem de reúso de teste. Ambos podem ser utilizados durante o uso do arcabouço de reengenharia definido no próximo capítulo.

Capítulo

3

ARA: Um Arcabouço de Reengenharia Ágil

3.1 Considerações Iniciais

O uso de padrões de software e frameworks está sendo cada vez mais intenso no desenvolvimento de software (Markiewicz e Lucena, 2001). Uma das preocupações da Engenharia de Software é migrar sistemas legados para novas tecnologias e paradigmas. Assim, este capítulo apresenta um arcabouço de reengenharia ágil que apóia a migração de sistemas legados procedimentais para sistemas baseados em padrões e frameworks, utilizando para isso o apoio de ferramentas, processos, abordagens de reúso e práticas de métodos ágeis. Este capítulo apresenta também um processo de reengenharia, que foi definido para viabilizar o uso de tal arcabouço, objetivando reduzir alguns dos riscos associados à reengenharia de software, como aqueles ressaltados por Rosenberg (1996) e apresentados na Seção 2.2 do Capítulo 2. Outros recursos também foram criados nesta tese para serem associados ao arcabouço e estão apresentados nos Capítulos 4, 5, 6.

O capítulo está organizado da seguinte forma: Na Seção 3.2 é apresentado o arcabouço de reengenharia ágil, denominado ARA. Na Seção 3.3 descreve-se o processo ágil de reengenharia, denominado PARFAIT (**P**rocesso **Á**gil de **R**eengenharia baseado em **F**r**A**mework no domínio de sistemas de **I**nformação com técnicas de VV&**T**), enfocando principalmente os objetivos das suas fases e atividades, e justificando a sua agilidade. Na Seção 3.4 comenta-se dois estudos de caso para avaliar a adequação das atividades do PARFAIT,

inicialmente definidas. Na Seção 3.5 apresenta-se a análise dos resultados de um estudo de caso que foi feito para avaliar a aplicabilidade e a efetividade desse processo. A partir dos resultados desses estudos de caso, foram identificadas algumas necessidades, comentadas na Seção 3.6, para que o processo e, conseqüentemente, o arcabouço, pudesse ser aplicado mais efetivamente. Na Seção 3.7 são apresentadas as considerações finais deste capítulo.

3.2 ARA: Arcabouço de Reengenharia Ágil

O **A**rcabouço de **R**eengenharia **Á**gil, denominado ARA, foi definido com o intuito de apoiar a migração de sistemas procedimentais para o paradigma orientado a objetos, fornecendo um produto com qualidade e produzido em um curto espaço de tempo. Para isso, utiliza abordagens de reúso existentes em diversos níveis de abstração; bem como processos, ferramentas e práticas de métodos ágeis no contexto de reengenharia, seguindo o modelo de processo incremental e iterativo (Pressman, 2005; Sommerville, 2000), como apresentado na Figura 3.1. Todos esses recursos colaboram para a redução do tempo, do custo e de alguns riscos de reengenharia, identificados por Rosenberg (1996). Além disso, este arcabouço complementa o conjunto de abordagens de reengenharia definidas por esse mesmo autor.

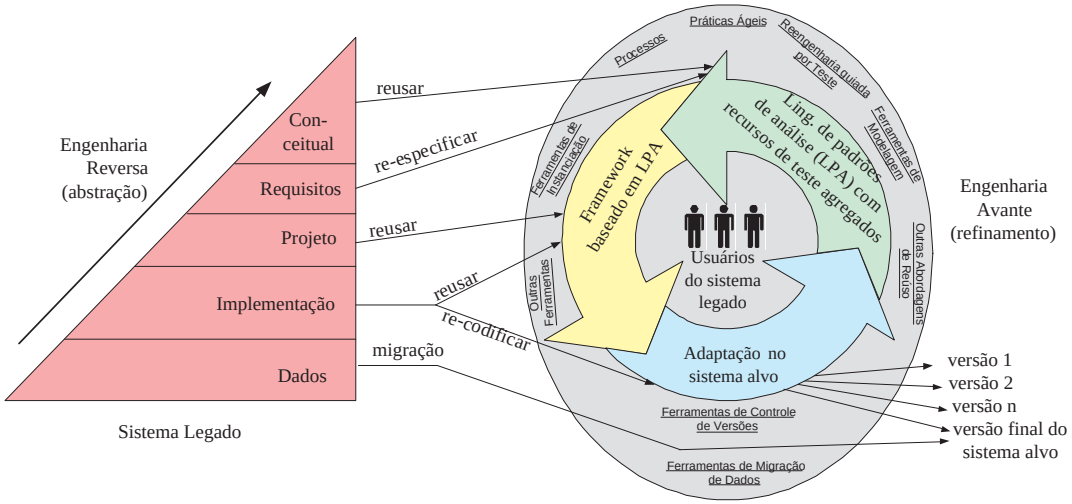


Figura 3.1: Abordagem ARA

Os conceitos, o entendimento e a análise do sistema legado são obtidos a partir de uma linguagem de padrões de análise com recursos de teste funcional agregados, pertencente ao mesmo domínio do sistema legado. O projeto e a implementação do sistema são reutilizados, em grande parte, do framework utilizado pelo arcabouço, cuja construção é baseada em uma linguagem de padrões de análise. Regras do negócio e requisitos específicos do sistema legado são implementados manualmente pelo engenheiro de software no sistema gerado a partir do framework (essa atividade é denominada adaptação). Então, o arcabouço não mantém componentes do sistema legado em conjunto com os do sistema alvo.

Como este arcabouço baseia-se na reengenharia incremental, a ordem de prioridade dos requisitos para serem submetidos à reengenharia é estabelecida pelos usuários [1] do sistema legado, que participam de todo o projeto de reengenharia e validam todos os artefatos produzidos à medida que o projeto evolui. Versões parciais do sistema alvo são demonstradas aos usuários concomitantemente com a versão do sistema legado para validá-las e garantir a equivalência funcional, até atingir a versão final do sistema alvo. Para garantir a qualidade do sistema alvo, testes são praticados desde o início da reengenharia, ou seja, todo o entendimento do sistema legado é baseado na execução de sua funcionalidade por meio de casos de teste reutilizados da linguagem de padrões, quando apropriado, e não por meio do código fonte. Esses casos de teste são posteriormente utilizados para validar o sistema alvo e também para garantir que possuem funcionalidade equivalente. Essa técnica é denominada "reengenharia guiada por teste".

O arcabouço ARA possui também controle de versão de todos os artefatos produzidos e permite que mudanças de requisitos sejam aceitas em qualquer momento da reengenharia. Antes de liberar a versão final do sistema para os usuários, a migração dos dados do sistema legado é feita para a base de dados do sistema alvo, com o apoio de ferramentas específicas para isso, e todos os casos teste previamente utilizados são novamente executados no sistema alvo para avaliar a sua confiabilidade e maturidade.

Além disso, o arcabouço agrega diversos benefícios de métodos ágeis, por exemplo, é adaptativo; tem participação constante dos usuários/clientes; é incremental, produzindo uma primeira versão do sistema alvo o mais rápido possível; utiliza testes desde o início da reengenharia; incentiva programação em pares; pratica jogo do planejamento a cada iteração do processo; garante propriedade coletiva do código, por meio do controle de versão de todos os artefatos produzidos durante a reengenharia; incentiva jornada de trabalho de no máximo quarenta horas semanais e garante a prática metáfora, por meio do uso da linguagem de padrões de análise. O uso de classes herdadas do framework aumenta a confiabilidade do sistema alvo uma vez que essas classes já foram testadas anteriormente.

3.3 Processo PARFAIT

PARFAIT (Cagnin et al., 2003c) é um processo que tem como objetivo migrar sistemas procedimentais para o paradigma orientado a objetos e, como mencionado no início deste capítulo, foi criado para ser associado ao arcabouço de reengenharia ágil definido. As principais características desse processo estão enumeradas a seguir:

- é incremental, iterativo e baseado em framework (atende os valores de métodos ágeis "resposta a mudanças" e "software operacional", comentados na Seção 2.5 do Capítulo 2);

[1] neste capítulo, a palavra usuário engloba também os clientes do sistema legado.

- considera diversas práticas de métodos ágeis (versões pequenas, cliente presente, testes constantes, jogo do planejamento, programação em pares, propriedade coletiva do código, integração contínua, metáfora e semana de 40 horas), atendendo todos os valores de métodos ágeis;

- é dirigido ao cliente e dirigido ao risco (atende os valores de métodos ágeis "pessoas e interações", "software operacional" e "participação contínua dos clientes");

- utiliza "reengenharia guiada por teste" (atende os valores de métodos ágeis "resposta a mudanças" e "software operacional");

- executa o sistema alvo concomitantemente com o sistema legado para avaliar a compatibilidade funcional entre eles (atende os valores de métodos ágeis "software operacional");

- não se limita a reproduzir a funcionalidade do sistema legado, mas evolui o sistema de acordo com as necessidades dos usuários (atende os valores de métodos ágeis "resposta a mudanças" e "software operacional"); e

- o formato da documentação é baseado em diversos elementos fundamentais do arcabouço do RUP (Kruchten, 2000).

A construção do framework utilizado como apoio computacional deve ter sido baseada em linguagem de padrões para facilitar o entendimento do sistema legado e permitir a elaboração de sua documentação OO, sendo que o seu domínio deve ser o mesmo que o do sistema legado. Isso pode minimizar os riscos de "elicitação de objetos incompleta e incorreta" e "dificuldade de recuperar o projeto e os requisitos a partir do código fonte", discutidos por Rosenberg (1996) e apresentados no Capítulo 2. Ressalta-se que não é necessário recuperar o projeto do sistema legado, pois é obtido diretamente do projeto da hierarquia de classes do framework.

Como mencionado no Capítulo 2, salienta-se que frameworks cuja construção é baseada em linguagem de padrões podem ser instanciados tomando como base o diagrama de classes do sistema desejado, construído a partir do uso dos padrões dessa linguagem, pois a funcionalidade de cada padrão é implementada pelas classes do framework.

Algumas características dos frameworks são herdadas pelos sistemas gerados a partir de sua instanciação. No caso de frameworks baseados em linguagem de padrões, eles permitem que os sistemas gerados herdem padrões de análise, padrões de projeto, código fonte na linguagem de implementação em que eles foram implementados, meio de armazenamento utilizado (por exemplo, SGBD relacional, objeto-relacional ou orientado a objetos), entre outros. No caso específico do framework GREN (Braga, 2003), que é baseado na linguagem de padrões GRN (Braga et al., 1999), há reúso dos padrões de análise da GRN, dos padrões de projeto (por exemplo, *Strategy*, *Factory Method*, *Template Method* (Gamma et al., 1995)

e *PersistenceLayer* (Yoder et al., 1998)) embutidos nas superclasses do framework, do código fonte do sistema na linguagem de implementação Smalltalk (Beck, 1997; Shafer, 1991) e das tabelas do sistema no SGBD MySQL (MySQL, 2003).

Como PARFAIT é iterativo e possui apoio computacional baseado em framework, fornece uma versão inicial e operacional do sistema alvo, que evolui de forma incremental durante a aplicação do processo até atingir a versão final. Isso minimiza também o risco de "elicitação de objetos incompleta e incorreta" (Rosenberg, 1996). Cada versão do sistema é construída de acordo com a prioridade dos requisitos do sistema legado em relação ao negócio, determinada pelos usuários. O processo permite também que o engenheiro de software retorne a qualquer uma de suas atividades em qualquer momento da reengenharia, objetivando refinar os artefatos produzidos de acordo com o *feedback* dos usuários. Essa característica do PARFAIT está de acordo com Larman (2004a), que afirma que passos curtos, rápido *feedback* e adaptação freqüente são as idéias centrais em desenvolvimento iterativo. Tudo isso colabora para o controle do risco de "insuficiência da engenharia reversa" (Rosenberg, 1996).

O processo conta também com a participação constante de usuários durante todo o projeto de reengenharia, para que eles possam avaliar continuamente cada um dos artefatos produzidos. Isso faz com que os riscos de "seleção inadequada de subsistemas submetidos ao processo de reengenharia", "ausência de garantia de qualidade do sistema resultante da reengenharia", "recuperação de informação sem utilidade ou não utilizada" e "dificuldade de recuperar o projeto e os requisitos a partir do código fonte", discutidos por Rosenberg (1996), sejam controlados.

As versões "intermediárias" do sistema alvo são liberadas para os usuários apenas para a finalidade de teste de aceitação. O sistema legado continua operando normalmente até a implantação do sistema alvo. Qualquer alteração de funcionalidade é comunicada aos responsáveis tanto da reengenharia quanto da manutenção do sistema legado (se houver), a qual é providenciada o mais rápido possível no sistema alvo.

Com relação à característica do PARFAIT "dirigido ao cliente", os usuários indicam quais os requisitos do sistema legado são mais importantes para o negócio e que devem ser primeiramente considerados na reengenharia. Com relação à característica "dirigido ao risco", PARFAIT concentra-se em criar uma arquitetura central e sólida o mais cedo possível, com o apoio do framework, para minimizar os riscos da reengenharia, uma vez que o objetivo inicial é fornecer rapidamente uma versão do sistema alvo que funcione adequadamente.

A característica "reengenharia guiada por teste" do PARFAIT foi inspirada na prática ágil "testes constantes", que é também conhecida como *test-driven-development* (TDD) (Beck, 2002; Crispin e House, 2003) e *eXtreme Testing* (Myers, 2004b). Nessa prática, o teste de unidade do código fonte é escrito antes do seu desenvolvimento.

A "reengenharia guiada por teste" apóia o entendimento do sistema legado e a identificação de regras do negócio embutidas no código fonte, bem como a validação do sistema alvo.

Os dois primeiros casos (apoio ao entendimento do sistema legado e à identificação de regras do negócio) tentam controlar os riscos de "perda do conhecimento do negócio embutido no código fonte" e "dificuldade de recuperar o projeto e os requisitos a partir do código fonte", discutidos por Rosenberg (1996). Enquanto que o último caso (apoio à validação do sistema alvo) tenta controlar o risco de "ausência de garantia de qualidade do sistema resultante da reengenharia" (Rosenberg, 1996).

O PARFAIT foi documentado utilizando diversos elementos fundamentais do arcabouço do RUP (Kruchten, 2000): fases, marcos de referência, atividades, passos, papéis, artefatos de entrada e de saída, disciplinas, gabaritos, diretrizes e guia de ferramenta de apoio computacional.

A documentação de cada atividade do PARFAIT é composta pelos seguintes itens: 1) papel que deve executá-la; 2) artefatos de entrada requeridos para iniciar a atividade; 3) artefatos de saída que são produzidos pela atividade; 4) passos detalhados 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 de apoio computacional, caso elas não sejam de conhecimento geral do pessoal envolvido na reengenharia; 7) inspeções no formato de *checklist*, cujo objetivo é verificar se os artefatos de saída foram produzidos adequadamente – isso controla o risco de "ausência de garantia de qualidade do sistema resultante da reengenharia" (Rosenberg, 1996); 8) gabaritos utilizados como base para a construção dos artefatos de saída; e 9) diretrizes associadas a algumas atividades ou passos, a fim de apoiarem sua execução em técnicas específicas (por exemplo, framework disponível, linguagem de padrões, linguagem de programação e SGBD).

Embora as fases do PARFAIT possuam o mesmo nome que as do RUP, o objetivo de cada uma é específico para o contexto de reengenharia. Além disso, as fases são utilizadas na documentação do PARFAIT para agrupar atividades com objetivos em comum. Na Figura 3.2 é apresentada a visão geral do processo PARFAIT, destacando as suas fases, suas atividades, com indicação de obrigatoriedade ou não de execução, e os seus passos.

As diretrizes para apoiar o uso de técnicas específicas durante a aplicação do processo e as inspeções a serem realizadas para validar os artefatos produzidos estão representadas na Figura 3.2 pelas letras D(iretriz) e I(nspeção), respectivamente, dentro de um círculo, e estão associadas a atividades ou passos.

A fase de Concepção contém atividades relacionadas à identificação do escopo e do domínio do sistema legado em relação ao framework disponível, observando os riscos associados, e também à elaboração do planejamento do projeto de reengenharia, que é baseado em projetos similares concluídos e/ou na experiência do responsável pela reengenharia e é atualizado a cada iteração das fases do processo. Esse planejamento não é muito detalhado, somente fornece uma idéia do tempo e custo que serão necessários para a realização da reengenharia. No entanto, objetiva controlar os riscos de "não cumprimento do prazo e custo inicialmente estimados" e "alto custo da reengenharia" (Rosenberg, 1996).

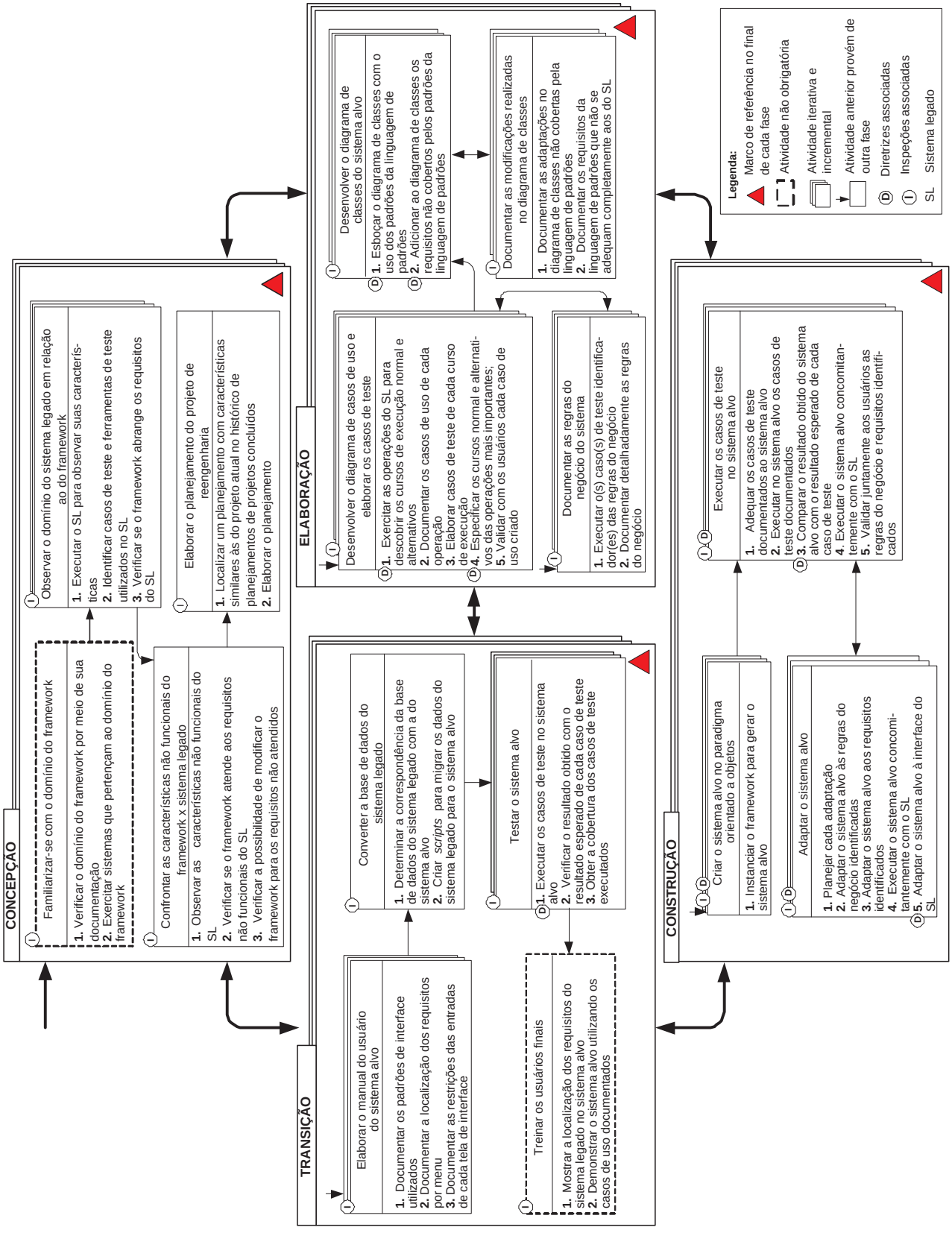


Figura 3.2: Visão Geral do PARFAIT

PARFAIT sugere que a reengenharia seja feita em pares e que o planejamento seja baseado em quarenta horas semanais, para que a sobrecarga de trabalho não afete a produtividade das pessoas envolvidas na reengenharia.

A fase de Elaboração possui atividades que preocupam-se com o entendimento do sistema legado baseando-se na execução do mesmo por meio de casos de teste; e com a elaboração da documentação OO do sistema legado, com o apoio da linguagem de padrões de análise, suficiente para instanciar o framework e para adaptar o sistema alvo produzido a partir dessa instanciação.

A fase de Construção possui atividades que fornecem uma versão do sistema alvo operacional o mais rápido possível a partir da instanciação do framework e apóiam a sua adaptação até atingir um sistema com funcionalidade semelhante à do sistema legado. A primeira versão do sistema alvo apóia o refinamento dos requisitos do sistema legado e a elicitação de eventuais novos requisitos, que podem ser fornecidos pelo framework e podem ser importantes para o negócio do sistema. Como o processo é iterativo, o framework pode ser instanciado várias vezes. Isso dependerá se os requisitos do legado selecionados para uma determinada iteração do processo são ou não fornecidos pelo framework.

A fase de Transição possui atividades que preparam o sistema alvo para ser implantado na empresa, que incluem elaborar o manual do usuário do sistema, realizar a conversão da base de dados legada para a do sistema alvo, avaliar a maturidade do sistema e treinar os usuários finais, caso seja necessário. Com relação a conversão da base de dados legada, PARFAIT motiva o uso de ferramentas existentes específicas para isso, para controlar o risco de "dificuldade na migração dos dados existentes" (Rosenberg, 1996).

No final de cada fase do processo há um marco de referência (ou ponto de checagem) que permite ao responsável pela reengenharia verificar o andamento do projeto e decidir pela sua continuidade ou não. Em cada marco de referência, o planejamento, que é realizado no final da fase de Concepção, deve ser revisitado e reajustado quanto ao tempo e aos esforços inicialmente previstos. Com isso, PARFAIT objetiva controlar os riscos de "não cumprimento do prazo e custo inicialmente estimados" (Rosenberg, 1996).

Após o término de cada iteração do PARFAIT, que pode ocorrer em qualquer uma das fases, os usuários indicam os próximos requisitos do sistema legado que devem ser considerados na iteração subseqüente.

Todos os artefatos produzidos durante um projeto de reengenharia são submetidos ao controle de versões e, a qualquer momento, é possível recuperar as versões produzidas. Isso pode ser apoiado por uma ferramenta específica, por exemplo, CVS (*Concurrent Versions System*) (Concurrent Versions System, 2004) e VersionWeb (Soares et al., 2000). Com isso, PARFAIT objetiva controlar parcialmente o risco de "gerenciamento de configuração inadequado" (Rosenberg, 1996).

A implantação do sistema alvo não é feita de forma incremental, porque o sistema legado está em execução e não pode parar. Uma solução poderia ser implantar o sistema alvo passo a passo e em paralelo ao sistema legado, mas isso gera custo extra para a empresa e, portanto, não foi adotada pelo PARFAIT. PARFAIT sugere que a implantação do sistema alvo seja feita depois do término da reengenharia.

As Tabelas de 3.1 a 3.4 apresentam um resumo do PARFAIT, com os objetivos de cada uma de suas fases e de suas atividades, os papéis responsáveis por cada atividade e os artefatos que devem ser produzidos.

As disciplinas do PARFAIT são apresentadas na Tabela 3.5, sendo que o objetivo de cada uma é alcançado a partir da execução de atividades distintas do processo. Graças à natureza iterativa e incremental do PARFAIT, a separação das etapas engenharia reversa e engenharia avante não é enfatizada nas disciplinas.

O processo PARFAIT exige, como pré-requisito, conhecimentos: 1) do paradigma OO; 2) do domínio do framework disponível; 3) da linguagem de padrões em que a construção do framework disponível foi baseada; 4) da linguagem de programação em que o framework foi construído, bem como do mecanismo que foi utilizado para o armazenamento dos dados do framework e dos sistemas gerados; 5) da linguagem UML (*Unified Modeling Language*) (Fowler e Scott, 1997; OMG, 2003), mais especificamente dos diagramas de casos de uso e de classes; e 6) dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite (Myers, 2004a). Espera-se que, quanto mais conhecimentos os engenheiros de software tiverem nesses itens, maior será o sucesso de aplicação do processo em um intervalo de tempo menor. Isso pode minimizar os riscos de "alto custo da reengenharia" e "ausência de conhecimento e experiência do pessoal envolvido na reengenharia" (Rosenberg, 1996).

Tabela 3.1: Resumo da fase de concepção do PARFAIT

Fase: **CONCEPÇÃO**
Objetivo: Observar os riscos de utilizar o framework disponível, baseado em uma linguagem de padrões de análise, na reengenharia do sistema legado.

Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos/atualizados**
Familiarizar-se com o domínio do framework (não obrigatória, caso o analista de sistema já possua familiaridade com o framework disponível)	Analisar e entender o domínio ao qual o framework pertence e verificar quais características inerentes a esse domínio são cobertas pelo framework.	Analista de sistemas	Formulário de avaliação do conhecimento do domínio do framework, Documentação de aceitação do framework.
Observar o domínio do sistema legado em relação ao do framework (obrigatória)	Identificar as características do sistema legado para verificar se ele pertence ao mesmo domínio do framework a fim de utilizá-lo no projeto de reengenharia. Essa atividade também identifica se existem casos de teste (para serem reutilizados no projeto de reengenharia) e ferramentas de teste (para obter informação sobre os critérios e coberturas de teste associados ao código legado), utilizados no desenvolvimento do sistema legado.	Analista de sistemas	Documentação de aceitação do framework, Documento de requisitos.
Confrontar as características não funcionais do framework x sistema legado (obrigatória)	Observar se as características não funcionais do framework são aceitáveis ou modificáveis para apoiar a reengenharia do sistema legado. O processo PREF (Capítulo 4) pode ser utilizado para apoiar tanto a decisão quanto a própria evolução do framework.	Analista de sistemas	Documentação de aceitação do framework, Documento de requisitos, Formulário para levantamento de requisitos não funcionais do sistema legado.
Elaborar o planejamento do projeto de reengenharia (obrigatória)	Observar planejamentos de projetos concluídos para elaborar o planejamento do projeto e para reduzir a possibilidade de superestimar ou subestimar o tempo e esforços envolvidos. Caso não haja planejamentos anteriores, a elaboração do planejamento é baseada na experiência do responsável pela reengenharia. Sugere-se que no planejamento seja determinada uma carga horária de, no máximo, quarenta horas semanais, com engenheiros de software trabalhando em pares, se possível.	Responsável do projeto de reengenharia.	Planejamento do projeto de reengenharia.

Marco de Referência: Critérios de avaliação são estabelecidos para: 1) verificar a viabilidade de uso do framework disponível na reengenharia do sistema legado e 2) para decidir se o projeto de reengenharia deve ou não continuar.

Tabela 3.2: Resumo da fase de elaboração do PARFAIT

Fase: **ELABORAÇÃO**
Objetivo: Elaborar a documentação do sistema legado para apoiar a fase de CONSTRUÇÃO, tanto na instanciação do framework quanto na adaptação do sistema alvo obtido a partir dessa instanciação.

Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos/atualizados**
Desenvolver o diagrama de casos de uso e elaborar os casos de teste (obrigatória)	Entender os requisitos do sistema legado, priorizados pelos usuários para a iteração corrente, por meio da execução de casos de teste criados/reutilizados, bem como aqueles identificados na atividade "Observar o domínio do sistema legado em relação ao do framework".	Analista de sistemas, Usuários.	Diagrama de casos de uso, Documentação dos casos de teste, Documentação das classes de equivalência, Documento de requisitos.
Desenvolver o diagrama de classes do sistema alvo (obrigatória)	Desenvolver o esboço do diagrama de classes dos requisitos do sistema legado, priorizados pelos usuários para a iteração corrente, tomando como base a linguagem de padrões, em que a construção do framework disponível foi baseada.	Analista de sistemas.	Diagrama de classes do sistema, Resumo dos padrões e variantes utilizados na construção do esboço do diagrama de classes.
Documentar as modificações realizadas no diagrama de classes (obrigatória)	Documentar os padrões utilizados no diagrama de classes, mas que não se adequam completamente aos requisitos do legado e também os elementos adicionados no diagrama de classes do sistema, que refletem os requisitos do sistema legado não cobertos pela linguagem de padrões.	Analista de sistemas.	Quadro das adaptações no diagrama de classes não cobertas pela linguagem de padrões, Quadro dos requisitos da linguagem de padrões que não se adequam completamente aos do sistema legado.
Documentar as regras do negócio do sistema (obrigatória)	Documentar cada regra do negócio identificada pelos casos de teste ou solicitadas pelos usuários do sistema legado.	Analista de sistemas	Documentação das regras do negócio, Diagrama de classes do sistema.

Marco de Referência: Critérios de avaliação são estabelecidos para verificar: 1) se é possível ter uma visão geral e adequada dos requisitos do sistema legado, priorizados pelos usuários para a iteração corrente; 2) se esses requisitos foram documentados e se todas as regras do negócio relacionadas a tais requisitos foram consideradas; e 3) se a utilização do framework é realmente viável. Além disso, é necessário revisitar o planejamento do projeto de reengenharia para ajustar os esforços e tempos, de acordo com os recursos despendidos e com aqueles que serão necessários para conduzir as próximas atividades do processo para a iteração corrente.

Tabela 3.3: Resumo da fase de construção do PARFAIT

Fase: **CONSTRUÇÃO**
Objetivo: Criar o sistema alvo no paradigma orientado a objetos, relacionado aos requisitos do sistema legado, priorizados pelos usuários para a iteração corrente, e adaptá-lo para torná-lo com funcionalidade compatível à do sistema legado.

Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos/atualizados**
Criar o sistema alvo no paradigma orientado a objetos (obrigatória)	Criar uma versão do sistema alvo no paradigma orientado a objetos por meio da instanciação do framework, de acordo com os requisitos priorizados pelos usuários para a iteração corrente. A instanciação do framework é baseada nos padrões da linguagem de padrões utilizados para criar/atualizar o diagrama de classes. Os requisitos e regras do negócio do sistema legado não fornecidos pelo framework devem ser registrados em um Histórico de Requisitos (Capítulo 4.2, página 96), para que possam ser analisados futuramente para serem ou não implementados no framework.	Programador	Sistema alvo.
Executar os casos de teste no sistema alvo (obrigatória)	Adequar e executar os casos de teste do sistema legado no sistema alvo para observar a diferença de comportamento entre as versões, visando descobrir regras do negócio e requisitos específicos do sistema legado ou presentes no framework e não requeridos pelo sistema legado. Executar o sistema alvo concomitantemente com o sistema legado, juntamente com os usuários, para elicitar novos requisitos ou refinar os existentes.	Testador, Usuários.	Subconjunto dos casos de teste identificadores de regras do negócio, Subconjunto dos casos de teste identificadores de requisitos, Documentação de casos de teste adequados ao sistema alvo.
Adaptar o sistema alvo (obrigatória)	Regras do negócio, requisitos não cobertos pela linguagem de padrões ou fornecidos pelo framework mas não requeridos pelo sistema legado, bem como sugestões dos usuários são consideradas como adaptações para serem realizadas no sistema alvo. Antes dessa tarefa os programadores planejam as adaptações para que cada uma seja finalizada em no máximo quinze dias. As sugestões dos usuários também devem ser registradas no Histórico de Requisitos.	Programadores (em pares, se possível), Usuários.	Planejamento das adaptações, Sistema alvo, Sumário dos comentários dos usuários sobre o sistema alvo.

Marco de Referência: Critérios de avaliação são estabelecidos para verificar: 1) se muitos requisitos e regras do negócio foram identificados com a execução dos casos de teste no sistema alvo, inviabilizando a sua adaptação; 2) se o sistema alvo poderá ser submetido às solicitações de mudanças sugeridas pelos usuários; e 3) se o sistema alvo está pronto para ser preparado para ser entregue aos usuários. Caso o sistema alvo não esteja pronto, é necessário retornar a atividade "Desenvolver o diagrama de casos de uso e elaborar os casos de teste" para que os usuários estabeleçam os requisitos para a próxima iteração do processo ou, se desejado, ir para a atividade "Elaborar o manual do usuário do sistema alvo" da próxima fase. Caso o sistema alvo esteja pronto, é necessário ir para a próxima fase. Além disso, é necessário revisitar o planejamento do projeto de reengenharia para ajustar os esforços e tempos, de acordo com os recursos despendidos e com aqueles que serão necessários para a iteração corrente do processo, ou para a próxima, se for o caso.

Tabela 3.4: Resumo da fase de transição do PARFAIT

Fase: **TRANSIÇÃO** *Objetivo*: Assegurar que o sistema alvo está pronto para ser disponibilizado para seus usuários e para ser implantado na empresa. A execução das atividades desta fase, com exceção da primeira, pode ser feita somente depois que não houver mais requisitos do legado para serem considerados na reengenharia.			
Atividades	**Objetivos**	**Papéis**	**Artefatos produzidos/atualizados**
Elaborar o manual do usuário do sistema (obrigatória)	Criar o manual do usuário do sistema alvo ou uma ajuda *on-line*.	Analista de Sistemas	Manual do usuário.
Converter a base de dados do sistema legado (obrigatória)	Migrar os dados da base de dados do sistema legado para a do sistema alvo.	Administrador de banco de dados	Base de dados atualizada do sistema alvo.
Testar o sistema alvo (obrigatória)	Submeter o sistema alvo aos casos de teste documentados para avaliar a maturidade do produto.	Testador	Sistema alvo liberado para uso.
Treinar os usuários finais (não obrigatória [2])	Capacitar os usuários a utilizar adequadamente o sistema alvo.	Analista de Sistemas	-
Marco de Referência: Critérios de avaliação são estabelecidos para avaliar: 1) a satisfação dos usuários e da empresa que solicitou a reengenharia; 2) o sucesso da conversão dos dados do legado para o sistema alvo orientado a objetos e 3) a garantia da qualidade desse sistema. Além disso, é necessário revisitar o planejamento do projeto de reengenharia para ajustar os esforços e tempos que serão necessários para a próxima iteração do processo, caso a reengenharia não tenha sido finalizada.			

Tabela 3.5: Atividades do PARFAIT para alcançar o objetivo das suas disciplinas

Disciplinas	**Atividades/Ações**
Entendimento do domínio do framework/sistema legado	Familiarizar-se com o domínio do framework, Observar o domínio do sistema legado em relação ao do framework, Confrontar as características não funcionais do framework x sistema legado.
Entendimento dos requisitos e da funcionalidade do sistema legado	Desenvolver o diagrama de casos de uso e elaborar os casos de teste.
Recuperação da análise e projeto	Desenvolver o diagrama de classes do sistema alvo, Documentar as modificações realizadas no diagrama de classes.
Implementação	Criar o sistema alvo no paradigma orientado a objetos, Adaptar o sistema alvo.
Teste & Inspeção	Desenvolver o diagrama de casos de uso e elaborar os casos de teste, Executar os casos de teste no sistema alvo, Testar o sistema alvo.
Modelagem de negócio	Executar os casos de teste no sistema alvo, Documentar as regras do negócio do sistema, Desenvolver o diagrama de classes do sistema alvo (atualização do diagrama de classes), Documentar as modificações realizadas no diagrama de classes.
Implantação	Elaborar o manual do usuário do sistema alvo, Converter a base de dados do sistema legado, Testar o sistema alvo, Treinar os usuários finais.
Gerenciamento de Configuração de Software	Aplicar controle de versão nos artefatos produzidos, no final da iteração de cada atividade.
Planejamento e Gerenciamento de Projeto	Elaborar o planejamento do projeto de reengenharia, marcos de referências (*milestones*) de cada fase.
Ambiente	Em cada atividade, ferramentas de apoio computacional são indicadas para facilitar a sua execução.

A documentação completa da atividade *Desenvolver o diagrama de casos de uso e elaborar os casos de teste* é apresentada a seguir. Essa atividade pertence à fase de elaboração e sua execução é obrigatória.

Atividade: Desenvolver o diagrama de casos de uso e elaborar os casos de teste (obrigatória)

Objetivo: Familiarizar o engenheiro de software com os requisitos do sistema legado, priorizados pelos seus usuários para a iteração corrente. Isso é feito por meio da execução do sistema legado, que é apoiada por casos de teste. O alcance dessa familiaridade refina a atividade "Observar o domínio do sistema legado em relação ao do framework" e permite que o artefato "Documento de requisitos" seja atualizado com a descrição dos requisitos funcionais do sistema legado, agrupados por interesse [3]. Os casos de teste são criados a partir de critérios de teste funcional ou por meio da abordagem ARTe, que é um dos produtos desta tese e é apresentada no Capítulo 5. Recomenda-se o uso dessa abordagem para reduzir o tempo das atividades de VV&T e, conseqüentemente, o custo da reengenharia. Assim, PARFAIT minimiza o risco de "alto custo da reengenharia", discutido por Rosenberg (1996). Os critérios de teste funcional foram selecionados pois visam testar a funcionalidade do sistema independentemente de uma implementação particular. Dentre os critérios da técnica funcional existentes, optou-se por utilizar os critérios Particionamento de Equivalência e Análise do Valor Limite, pois são os mais utilizados e difundidos na literatura. Caso existam muitos requisitos do legado para serem considerados na iteração corrente, esses podem ser distribuídos entre os vários analistas de sistemas e, no final da atividade, todos se reúnem para juntar os diagramas de casos de uso criados com seus casos de teste correspondentes.

Papéis: Analista de Sistemas, Usuários.

Artefatos de Entrada: Sistema legado, Documento de requisitos.

Apoio Computacional: Ferramenta CASE que apóia modelagem em UML (por exemplo, Rational Rose (IBM Rational Software, 2003) e ArgoUML [4]).

Guia da Ferramenta de Apoio: Para documentar os casos de uso com seus respectivos casos de teste é necessário:

1) criar um diagrama de casos de uso;

2) para cada operação de menu do sistema legado, criar um caso de uso (nome do caso de uso = identificador do caso de uso [5] + nome da operação do sistema legado [6]) e;

[3] Grupo de requisitos que possui interdependência funcional ou não funcional.
[4] `http://argouml.tigris.org/`
[5] UC1, UC2, UCn
[6] Verbo no infinitivo + complemento

3) para cada dado utilizado como entrada, para executar o caso de uso, e a saída obtida, após a execução do caso de uso, documentá-los como casos de teste e nomeá-los da seguinte maneira CT1, CT2, ..., CTn, Tabela 3.7.

Gabarito: Na Tabela 3.6 e na Tabela 3.7 são apresentados os gabaritos dos artefatos que devem ser construídos nesta atividade. Se o caso de teste é derivado do critério Particionamento de Equivalência, o número da classe de equivalência deve ser documentado (Tabela 3.7, terceira coluna).

Tabela 3.6: Documentação de Classes de Equivalência (Myers, 2004a)

Restrições de Entrada	**Classes Válidas**	**Classes Inválidas**
Conteúdo da senha do usuário	Seqüência de caracteres alfanuméricos **(1)**	Seqüência de caracteres numéricos **(2)**
...	...	...

Tabela 3.7: Documentação dos casos de teste

Id. Caso de Uso	**Id. Caso de Teste**	**Id. Classe Equivalência**	**Entrada**	**Saída Esperada**
UC1	CT1	1	<Dados entrada>	<Dados saída>
UC1	CT2	–	<Dados entrada>	<Dados saída>
UC2	CTn	2	<Dados entrada>	<Dados saída>

Inspeções:

I1. Quanto ao diagrama de casos de uso: verificar se existe um caso de uso para cada operação do sistema legado. Além disso, verificar se todos os relacionamentos de associação de entrada e saída estão documentados corretamente, ou seja, se não houve omissão, duplicação ou inconsistência nos relacionamentos.

I2. Quanto à documentação dos casos de teste: verificar se cada caso de teste possui um dado de entrada e um dado de saída correspondente.

I3. Quanto ao documento de requisitos: verificar se todos os requisitos funcionais e não funcionais da iteração corrente foram documentados adequadamente, se refletem as necessidades dos usuários e se foram agrupados adequadamente em seus respectivos interesses.

Artefatos de Saída: Diagrama de casos de uso, Documentação dos casos de teste, Documentação das classes de equivalência, Documento de requisitos.

Passos:

1. **Exercitar as operações do sistema legado para descobrir os cursos de execução normal e alternativos:** Este passo deve ser realizado em paralelo com os passos 4 e 5 juntamente com os usuários a fim de descobrir: operações que estão em desuso por motivos de mudanças na política econômica ou organizacional (essas não

devem ser consideradas no processo de reengenharia); operações que não satisfazem completamente às necessidades dos usuários; requisitos/regras do negócio relevantes que não estão implementados no sistema legado. Neste passo, o artefato "Documento de requisitos" deve ser atualizado, descrevendo os requisitos funcionais do sistema que estão sendo considerados na iteração corrente, agrupando-os por interesse (por exemplo: Interesse Cliente - contém todos os requisitos sobre o cliente; Interesse Venda - contém todos os requisitos que envolvem venda). Caso a implementação de alguma regra do negócio tenha sido solicitada pelos usuários é recomendável que seja documentada na atividade "Documentar as regras do negócio do sistema". Na primeira iteração deste passo, os usuários devem indicar os requisitos do sistema legado que são mais importantes para o negócio e que devem ser primeiramente considerados na reengenharia. Os demais requisitos são distribuídos nas iterações seguintes.
Diretrizes do Passo: Sugestão de perguntas a serem feitas aos usuários para identificar operações em desuso, incompletas ou requisitos não implementados.

D1. Quais operações não são mais utilizadas ou são utilizadas com pouca freqüência no sistema?

D2. Por que determinadas operações não são mais utilizadas? Elas não satisfazem às necessidades dos usuários ou não são mais utilizadas por motivos de mudanças na política econômica ou organizacional?

D3. Existem operações que são pouco utilizadas? Por quê? Elas são importantes e imprescindíveis, havendo a necessidade de mantê-las no sistema?

D4. Quais operações poderiam ser melhoradas para satisfazer completamente as expectativas da empresa? Solicitar sugestões de melhorias.

D5. Existem alguns requisitos que são executados manualmente na empresa? Quais? Há possibilidade de integrá-los ao sistema na reengenharia (isso será possível se esses requisitos estiverem inseridos no escopo do sistema)?

2. **Documentar os casos de uso de cada operação:** veja item *Guia da ferramenta de apoio*.

3. **Especificar os cursos normal e alternativos das operações mais importantes.**

4. **Elaborar e documentar casos de teste de cada curso de execução.** Para isso é necessário derivar casos de teste aplicando critérios de teste funcional. Inicialmente aplica-se o critério Particionamento de Equivalência e, em seguida, o critério de Análise do Valor Limite. Posteriormente, é necessário gerar para cada classe de equivalência casos de teste que estejam nos limites dessas classes e documentá-los de acordo com a Tabela 3.7. Neste passo, o testador deve documentar também, caso existam, os casos de teste utilizados durante o desenvolvimento do sistema legado (*CTdesenvol.*) e os

casos de teste típicos (*CTtipico*), ou seja, aqueles derivados das tarefas rotineiras dos usuários e catalogados na documentação burocrática da empresa (fichas de controle interno utilizadas antes da implantação do sistema legado) ou consultados a partir da base de dados do legado. Esses dois tipos de casos de teste foram recuperados durante o passo "Identificar casos de teste e ferramentas de teste utilizados no sistema legado" da atividade "Observar o domínio do sistema legado em relação ao do framework" (Fase de Concepção).

Diretrizes do Passo: Utilizar as diretrizes de reúso da abordagem ARTe que é apresentada no Capítulo 5 para executar este passo. Por meio dessa abordagem é possível reutilizar casos de teste, requisitos de teste e classes de equivalência previamente criados para o domínio da linguagem de padrões em que a construção do framework disponível na reengenharia foi baseada.

5. **Validar com os usuários cada caso de uso criado.**

Fim da Atividade

De acordo com Larman (2004a), as características de métodos ágeis são facilmente introduzidas em processos iterativos. Isso também pode ser observado no PARFAIT, que é um processo ágil, pois considera, de alguma maneira, diversas características e práticas ágeis (Abrahamsson et al., 2002; Beck, 2000). As práticas de XP (Beck, 2000) atualmente consideradas pelo PARFAIT são:

- *versões pequenas*: fornecidas a cada iteração do processo. No entanto, são liberadas para os usuários apenas para a finalidade de teste de aceitação;
- *cliente presente*: usuários participam ativamente da maioria das atividades do projeto de reengenharia e aprovam o produto à medida que o projeto evolui, sendo que a cada participação dos usuários, melhorias no produto são realizadas (isso facilita a satisfação dos usuários com a qualidade do produto final);
- *testes constantes*: testes funcionais utilizados para entender o sistema legado são executados no sistema alvo e os resultados são comparados visando a elicitação de requisitos e de regras de negócio, bem como a validação do sistema alvo. Testes de aceitação são realizados pelos usuários para aprovar o produto final;
- *jogo do planejamento*: o planejamento da reengenharia é ajustado a cada iteração em cada fase do PARFAIT;
- *programação em pares*: incentivada durante a adaptação do sistema alvo, mas é opcional;

- *propriedade coletiva do código e integração contínua*: qualquer programador pode alterar qualquer parte do código do sistema alvo quando for necessário - pois todas as versões de cada artefato produzido são submetidas ao controle de versões - e em seguida deve integrá-lo. Essas práticas podem ser restritas ou impraticáveis devido a algumas características inerentes aos frameworks e que podem não ter suporte das ferramentas de controle de versão existentes;

- *metáfora*: o entendimento do sistema legado é fornecido pelos padrões da linguagem de padrões, com base na qual o framework utilizado na reengenharia foi construído;

- *semana de 40 horas*: incentivada durante a elaboração do planejamento da reengenharia.

Ressalta-se que PARFAIT não restringe o uso das demais práticas de XP (ou seja, *refatoração constante*, *padrão de codificação* e *projeto simples*) e nem de outras práticas ágeis. O responsável pela reengenharia deve decidir por utilizá-las ou não. Caso as utilize, é necessário estabelecer políticas técnicas e de pessoal para viabilizá-las.

Todas as características de métodos ágeis, definidas por Abrahamsson et al. (2002), também são consideradas pelo PARFAIT: 1) *incremental*, isto é, a documentação, o entendimento do sistema legado e o sistema alvo são obtidos aos poucos; 2) *cooperativo*, pois conta com a opinião e a aceitação dos usuários; 3) *direto*, pois o processo é fácil de entender e possui uma documentação completa em forma de tutorial; no entanto, durante a reengenharia, elabora-se pouca documentação, suficiente para apoiar a migração do sistema legado para o sistema alvo e 4) *adaptável*, pois pode haver mudanças a qualquer momento durante a aplicação do processo e o engenheiro de software pode retornar a qualquer atividade para melhorar o produto que está sendo criado.

Outra preocupação abordada pelo PARFAIT é a Modelagem Ágil (MA) (Ambler e Jeffries, 2002), que tem como principal objetivo modelar e documentar o sistema legado de maneira efetiva e ágil, para promover o seu entendimento e para facilitar a comunicação entre os engenheiros de software. Para isso, PARFAIT: 1) fornece gabaritos de todos os artefatos que devem ser produzidos durante a sua aplicação para diminuir o tempo de criação da documentação; 2) incentiva os usuários a participarem da validação dos artefatos; 3) permite que os requisitos sejam alterados, conforme a necessidade; 4) motiva a criação de modelos simples e fáceis de serem entendidos; 5) faz com que a documentação do sistema seja construída de forma iterativa e incremental, conforme a necessidade e as prioridades determinadas pelo usuário; e 6) estabelece a criação de diversos tipos de modelos para expressar cada particularidade do sistema que está sendo submetido à reengenharia. Alguns artefatos são opcionais e somente devem ser criados de acordo com as necessidades do projeto. Tudo isso pode controlar o risco de "grande extensão e alto custo da documentação produzida", discutido por (Rosenberg, 1996).

PARFAIT utiliza apenas alguns diagramas da UML 1.x durante a elaboração da documentação do sistema alvo. Ressalta-se que a especificação da UML 1.4 foi evoluída para a versão 2.0 (Miller, 2003; Selic, 2004) para incorporar um nível maior de semântica aos diagramas, a fim de apoiar uma nova abordagem de desenvolvimento de software dirigida a modelo [7], denominada MDA (*Model Driven Architecture*) (J. Siegel and the OMG Staff Strategy Group, 2001; Miller e Mukerji, 2001; R. Soley and the OMG Staff Strategy Group, 2000). O objetivo dessa nova abordagem é permitir que o engenheiro de software especifique a estrutura e o comportamento do sistema em um modelo independente de plataforma e, em seguida, faça o mapeamento desse modelo para um modelo específico da plataforma em que o sistema será desenvolvido. A partir desse modelo, e com o apoio de ferramentas computacionais gera-se o código fonte do novo sistema. A nova versão da UML pode ser utilizada pelo PARFAIT, no entanto, como o PARFAIT preocupa-se com modelagem ágil que, de acordo com Larman (2004a), é um dos pontos-chaves para aplicar UML de maneira efetiva, recomenda-se que detalhes semânticos da nova versão da UML sejam evitados durante a construção dos diagramas na aplicação do PARFAIT.

De acordo com Larman (2004a), existem três perspectivas na aplicação da UML: 1) perspectiva conceitual, 2) perspectiva de especificação e, 3) perspectiva de implementação. Na primeira perspectiva os diagramas descrevem conceitos do mundo real ou do domínio de interesse, não contendo especificação de análise detalhada. Na segunda perspectiva, os diagramas descrevem abstrações do software ou componentes com especificações e interface, mas não possuem detalhes específicos de implementação em uma determinada linguagem de programação. Esses detalhes são tratados pela terceira perspectiva. PARFAIT utiliza apenas as duas primeiras perspectivas, sendo que a primeira é garantida pelo uso da linguagem de padrões na qual o framework foi construído, que apóia o entendimento do sistema legado e a elaboração do seu diagrama de classes, sem detalhes específicos de linguagem de programação. A criação do diagrama de classes e a do diagrama de casos de uso do sistema legado, fazem com que PARFAIT utilize a segunda perspectiva de uso da UML.

Do ponto de vista de experimentação, um planejamento de pacote de laboratório, apresentado no Apêndice A, foi criado especificamente no contexto de reengenharia, para observar a aplicabilidade do PARFAIT em relação a outros processos de reengenharia "ad hoc". Esse planejamento foi parcialmente validado por meio de um estudo de caso. O pacote de laboratório contém toda a instrumentação necessária (ou seja, documentos e material de treinamento) para apoiar os interessados que desejam conduzir tal experimento, tanto no meio industrial quanto no meio acadêmico. Com a replicação do experimento será possível, além de completar o pacote, refiná-lo e também melhorar a qualidade do processo PARFAIT

[7] Nesse tipo de abordagem, o foco do desenvolvimento são os modelos/diagramas, por isso são considerados como os artefatos principais. Tais modelos/diagramas são representados em uma linguagem de modelagem, por exemplo, UML (Selic, 2004).

à medida que for utilizado por outras pessoas e em outros ambientes com cultura diferente daquela em que já foi aplicado.

3.4 Estudos de Caso para Refinar o PARFAIT

Nesta seção apresentam-se alguns dos resultados de dois estudos de caso (Cagnin, 2002a,b), para observar a adequação das atividades do PARFAIT inicialmente definidas. Para isso, a primeira versão do processo foi utilizada em dois estudos de caso de reengenharia: de um **sistema de controle de biblioteca**, que controla o empréstimo e a devolução de livros, desenvolvido em Clipper, com aproximadamente 6 KLOC; e de um **sistema de controle de oficina eletrônica**, que controla a entrada e saída de aparelhos eletrônicos para manutenção, também desenvolvido em Clipper, com aproximadamente 4,5 KLOC. Esses estudos foram conduzidos pela autora desta tese.

Como o objetivo desses estudos de caso foi apenas aplicar o processo para observar a sua viabilidade de uso, nenhum tipo de coleta de dados foi realizada. Durante os estudos de caso, diversas inconsistências foram encontradas nas atividades do processo. A descrição e os passos das atividades foram melhor definidos para facilitar o uso do processo e os nomes de algumas atividades foram alterados para nomes mais significativos. Algumas atividades foram realocadas para outras fases, diferente daquelas que tinham sido inicialmente determinadas. Após o término dos estudos de caso, os objetivos inicialmente definidos para cada fase do processo foram refinados.

Um fator importante observado durante os estudos de caso foi em relação à cobertura da funcionalidade dos sistemas legados pelo framework GREN. Apesar de pertencerem ao mesmo domínio, alguns requisitos dos sistemas legados não eram fornecidos pelo GREN. Por exemplo:

1. **Sistema de Biblioteca:**

   (a) um livro possui diversos autores e assuntos (atributos que possuem mais do que um valor). Com o framework GREN, foi possível obter diretamente apenas um autor/assunto para o livro, por meio de uma classe herdada da superclasse `SimpleType`, que implementa o participante "Tipo do Recurso" do padrão "IDENTIFICAR O RECURSO" da linguagem de padrões GRN.

   (b) um aluno está matriculado em um curso (atributo cujo conteúdo é uma referência a outra classe). Com o GREN foi possível obter diretamente apenas o curso ao qual o aluno pertence, por meio de um atributo na classe Aluno, gerando redundâncias e inconsistências, pois em cada inserção de aluno o usuário terá que digitar o nome do curso em que o aluno está matriculado. Isso porque o GREN não possui

implementação de relacionamentos de outras classes com a classe `Resource` que, neste caso, é superclasse da classe `Aluno`.

(c) um aluno possui um determinado estado civil (solteiro, casado, divorciado, etc) e também uma determinada situação (ativo, inativo, desistente); a procedência de um livro pode ser estabelecida a partir de uma doação ou de uma compra; um livro pertence a uma determinada área (Exatas, Biológicas, Humanas) e pode estar ou não liberado para empréstimo. Esses são atributos cujos valores são determinados a partir de um conjunto de informações pré-determinadas. O GREN não possui nenhum recurso para facilitar a implementação desse tipo de informação.

2. **Sistema de Oficina Eletrônica:**

(a) cada aparelho eletrônico a ser consertado possui um proprietário. Isso pôde ser diretamente obtido do GREN apenas como um atributo da classe `Aparelho`, gerando redundâncias e inconsistências, similarmente ao item 1 (b).

(b) alguns dados do sistema da oficina eletrônica são obtidos a partir de um conjunto de informações pré-determinadas. Por exemplo: A cidade em que um cliente mora está localizada em uma determinada unidade federativa (SP, MG, RJ, ES, etc). Isso não pôde ser obtido diretamente do GREN pois não possui esse tipo de informação implementado, similarmente ao item 1 (c).

(c) o sistema possui diversos relatórios para a emissão de etiquetas. A versão original do framework GREN não possui nenhuma classe que fornece esse formato de relatório para ser reutilizado.

A partir disso, observou-se que os requisitos não cobertos pelo GREN estão relacionados a atributo enumerado com valor a partir de um objeto de uma classe (itens 1.b e 2.a), atributo enumerado com valor a partir de valores pré-definidos (itens 1.c e 2.b), atributo multivalorado com valores a partir de objetos de uma classe (item 1.a), e emissão de relatórios em formato de etiquetas (item 2.c), e são inerentes à maioria dos sistemas pertencentes ao domínio de Gestão de Recursos de Negócios. A ausência desses requisitos pode limitar o uso do framework GREN na construção de sistemas com essas características. Dessa forma, decidiu-se introduzi-los nesse framework para torná-lo mais flexível, permitindo que seja utilizado na reengenharia e no desenvolvimento de um número maior de sistemas desse domínio. A evolução do GREN foi feita com o apoio do processo de evolução de frameworks de aplicação, que é apresentado no Capítulo 4.

Na próxima seção apresenta-se um resumo do estudo de caso de reengenharia (Cagnin et al., 2003a; Chan et al., 2003) utilizando o PARFAIT, documentado segundo Wholin et al. (2000). Esse estudo permitiu a avaliação e o refinamento do processo PARFAIT e a observação de necessidades importantes, por exemplo, relacionadas a recursos e mecanismos

computacionais que poderiam colaborar para o sucesso da aplicação deste processo e, conseqüentemente, do arcabouço de reengenharia ágil definido.

3.5 Um Estudo de Caso para Avaliar o PARFAIT

Definição

Objeto de Estudo: PARFAIT.

Propósito: Estudo de caso para avaliar a aplicação do processo PARFAIT em um projeto de reengenharia.

Foco qualitativo: Efetividade do PARFAIT em relação ao apoio na reengenharia de sistemas procedimentais para o paradigma OO.

Perspectiva: A perspectiva é em relação a engenheiros de software interessados na reengenharia de sistemas no domínio de Gestão de Recursos de Negócios.

Contexto: O estudo de caso foi realizado por uma aluna de iniciação científica (IC), que cursava o início do quarto ano do curso de Bacharelado em Ciência da Computação no ICMC-USP, e teve como material básico a documentação do tutorial do PARFAIT, refinada a partir da experiência obtida com os dois estudos de caso discutidos na Seção 3.4, a linguagem de padrões GRN, a documentação da ferramenta de instanciação GREN-Wizard (Braga, 2003; Braga e Masiero, 2003) e um material de treinamento da linguagem Smalltalk, do SGBD MySQL e dos critérios de teste Particionamento de Equivalência e Análise do Valor Limite. O estudo é classificado como objeto único (ou seja, um objeto e um participante) e foi iniciado no mês de fevereiro de 2003. O período inicialmente estimado para a conclusão do estudo de caso foi de sessenta dias.

Planejamento

Seleção do Contexto: O estudo de caso foi conduzido de forma independente, sendo que a aluna de IC teve a liberdade de estipular seus próprios horários para a realização das atividades do PARFAIT, desde que o controle dos horários de início e fim dessas atividades fosse anotado em uma planilha fornecida. A aluna de IC conhecia pouco a linguagem de padrões GRN e já havia utilizado a ferramenta GREN-Wizard. O sistema legado que foi utilizado no estudo de caso é de pequeno porte, foi desenvolvido em Clipper e estava em uso no momento da condução do estudo. Esse sistema legado controla o empréstimo e a devolução de livros de uma biblioteca universitária e possui aproximadamente 6 KLOC e é um dos utilizados em um dos estudos de caso comentados na Seção 3.4. Esse estudo é válido em um contexto específico do domínio de Engenharia de Software.

Formulação das Hipóteses

Hipóteses nulas:

- H01: A minoria dos requisitos do sistema legado estão presentes no sistema alvo resultante da instanciação do framework.
- H02: O reúso de linhas de código fonte do framework é baixo.
- H03: O apoio de casos de teste para identificar regras do negócio do sistema legado é baixo.

Hipóteses alternativas:

- HA1: A maioria dos requisitos do sistema legado estão presentes no sistema alvo resultante da instanciação do framework.
- HA2: O reúso de linhas de código fonte do framework é alto.
- HA3: O apoio de casos de teste para identificar regras do negócio do sistema legado é alto.

Seleção das variáveis

Variáveis independentes [8]**:** A *experiência* do participante no paradigma OO, na linguagem de padrões GRN, no framework GREN e no domínio de Gestão de Recursos de Negócios; o *nível de escolaridade* e a *especialidade* do participante nas áreas de computação são fatores que influenciam o uso do PARFAIT. Além desses fatores, há também aqueles relacionados às *características do sistema legado*, ou seja, domínio e quantidade de linhas de código fonte.

Variáveis dependentes [9]**:** quantidade total de requisitos do sistema legado, quantidade de requisitos do sistema legado não encontrados na primeira versão do sistema alvo; porcentagem de linhas de código herdadas do framework (quantidade total de linhas de código da primeira versão do sistema alvo/quantidade total de linhas de código da versão final do sistema alvo), quantidade de regras do negócio do sistema legado identificadas pelos casos de teste, quantidade de regras de negócio do sistema legado não identificadas pelos casos. As duas primeiras variáveis estão relacionadas à cobertura do domínio do framework em relação ao do sistema legado de maneira intrínseca, ou seja, quanto mais relacionado ao domínio do framework estiver o domínio do sistema legado, menor será a quantidade de requisitos do sistema legado não encontrados na primeira versão do sistema alvo e maior será a porcentagem de linhas de código herdadas do framework. As duas últimas variáveis estão relacionadas à complexidade do sistema legado em relação à implementação das regras do negócio da empresa.

Seleção dos indivíduos: A técnica de escolha foi a amostragem por conveniência, uma vez que o participante era uma aluna de IC cujo projeto estava inserido no do PARFAIT. Um

[8]São variáveis que podem ser manipuladas e controladas.
[9]São variáveis nas quais se observa o efeito das mudanças das variáveis independentes.

dos usuários do sistema legado participou durante toda a condução do estudo para validar os artefatos produzidos.

Projeto do estudo de caso: O estudo de caso foi realizado por um participante em apenas um projeto de reengenharia.

Instrumentação: Documento do processo PARFAIT, linguagem de padrões GRN, formulário para coleta de dados do estudo de caso, manual da ferramenta de instanciação GREN-Wizard, documento de treinamento da linguagem de programação Smalltalk, do SGBD MySQL, e dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite, sistema legado executável, framework GREN e ferramenta de instanciação GREN-Wizard.

Avaliação da validade: *Validade de conclusão:* alguns dados coletados são subjetivos e dependem da qualificação do engenheiro de software que está utilizando o processo; *Validade interna:* o estudo de caso foi realizado pela aluna de IC sem restrição de dia, horário e local, portanto ela pôde conduzí-lo no momento que achou mais adequado. A aluna de IC, em uma outra ocasião, já utilizou uma versão anterior do processo PARFAIT com algumas atividades semelhantes à sua versão atual, portanto, como já teve uma experiência anterior com esse processo o seu desempenho pode ser melhor do que se fosse a primeira vez. A aluna de IC não era voluntária, pois era aluna de IC do processo PARFAIT. Isso pode influenciar os resultados; *Validade de construção:* como o estudo de caso foi realizado somente com um participante e com um objeto, muito provavelmente não dará a visão completa do que se pretende provar; *Validade externa:* esse estudo de caso foi conduzido no meio acadêmico, mas em um sistema legado real.

Operação

Execução: O estudo de caso foi conduzido em duas etapas. A primeira etapa consistiu no treinamento e no estudo das técnicas necessárias para o início do estudo de caso: 18 horas para o aprendizado da documentação do PARFAIT, 6 horas para o aprendizado da linguagem de padrões GRN, 20 horas para o treinamento da ferramenta de instanciação GREN-Wizard, da linguagem de programação Smalltalk e do SGBD MySQL, 4 horas para o treinamento dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite. A segunda etapa consistiu na aplicação do PARFAIT durante a reengenharia do sistema de controle de biblioteca. Durante essa etapa a aluna de IC registrou a seqüência da execução das atividades do PARFAIT, o tempo gasto em cada atividade, o tempo gasto na inspeção dos artefatos produzidos em cada atividade e o número de erros encontrados durante a inspeção em cada atividade, conforme apresentado na Tabela 3.8. Nessa tabela apresenta-se também o tempo planejado para realizar cada atividade, estimado na atividade "Elaborar o planejamento do projeto de reengenharia". Para os marcos de referência, que ocorrem no final de cada iteração das fases do PARFAIT, estimou-se um tempo total de 4 horas. O tempo para

realizar as inspeções nos artefatos foi estimado para cada atividade do PARFAIT, totalizando 30 horas, mas não está apresentado na Tabela 3.8. O tempo gasto nas atividades com nome em itálico e o tempo gasto nas inspeções dessas atividades não é computado no tempo total, pois essas atividades foram conduzidas antes do planejamento. Ressalta-se que os tempos estimados na atividade "Elaborar o planejamento do projeto de reengenharia" foram baseados na experiência da aluna de IC, pois não havia projetos concluídos de reengenharia para ela se embasar. Os casos de teste foram implementados e executados com o apoio do framework de teste S-Unit. A aluna não registrou o tempo exato despendido nos marcos de referência no final da iteração de cada fase, mas afirmou que gastou, no máximo, quinze minutos em cada marco. A totalização do tempo gasto nos quatro marcos de referência (uma hora) está computada no tempo total da reengenharia, apresentado na Tabela 3.8.

Outros dados foram coletados durante a condução do estudo de caso e são referentes as variáveis dependentes, descritas anteriormente, e estão apresentados na Tabela 3.9. Esses dados apoiaram na análise das hipóteses definidas. Nenhuma regra do negócio foi identificada pelos casos de teste, pois o sistema legado é simples e somente possui funcionalidades básicas implementadas.

Conforme relatado pela aluna de IC, no final da primeira execução da atividade "Executar os casos de teste no sistema alvo", o usuário do sistema legado sugeriu a implementação de uma regra do negócio (RN1) relacionada ao período autorizado para o usuário permanecer com o livro. Esse período é diferente para professores (sete dias) e outros usuários da biblioteca (três dias). A implementação de outra regra do negócio (RN2) foi solicitada durante a demonstração do sistema OO no final da primeira execução da atividade "Adaptar o sistema alvo", executando-o concomitantemente com o sistema legado. A RN2 é relacionada a cobrança de taxa de multa (R$ 1,00) quando o usuário da biblioteca retorna o livro depois da data estipulada para devolução.

Salienta-se que apesar do PARFAIT sugerir a reengenharia em pares, o estudo de caso não observa tal aspecto, pois foi conduzido apenas por um participante.

Análise e Interpretação dos Resultados

De acordo com a Tabela 3.8, foi gasto um tempo menor (8:37 horas) para inspecionar os artefatos elaborados do que o estimado (30 horas) no artefato *Planejamento da Reengenharia*, criado na atividade "Elaborar o planejamento do projeto de reengenharia". Esse artefato foi criado apenas a partir da experiência da aluna de IC, pois não havia planejamento de projetos anteriores de reengenharia em que se pudesse embasar. Infere-se que a razão do tempo gasto ser menor do que o estimado é devido ao uso de inspeções, em forma de *checklists*, fornecidas pelo PARFAIT. Poucos erros foram encontrados durante essas inspeções.

Além disso, observou-se que a maior parte do tempo foi gasta para executar a atividade "Desenvolver um diagrama de casos de uso e documentar os casos de teste". Do tempo total

Tabela 3.8: Coleta de dados do estudo de caso para avaliar o PARFAIT (Cagnin et al., 2003b)

Seqüência das atividades executadas	Atividades do PARFAIT	Tempo estimado por atividade	Tempo gasto por atividade	Tempo gasto na inspeção	Qtde erros encontrados na inspeção
1	*Familiarizar-se com o domínio do framework*	–	2:00	0:07	0
2	*Observar o domínio do sistema legado em relação ao do framework*	–	0:40	0:15	0
3	*Confrontar as características não funcionais do framework com as do sistema legado*	–	0:50	0:15	0
4	*Elaborar o planejamento do projeto de reengenharia*	–	1:00	0:00	0
5,18,25	Desenvolver o diagrama de casos de uso e documentar os casos de teste	119:00	552:50	2:04	5
6,8,10,15,22	Desenvolver o diagrama de classes do sistema alvo	13:00	16:10	2:34	4
7,9,11,16,23	Documentar as modificações realizadas no diagrama de classes	5:00	8:16	0:51	0
14,21	Documentar as regras do negócio do sistema	11:00	1:15	0:12	0
12	Criar o sistema alvo no paradigma orientado a objetos	8:30	14:10	0:15	0
13,19,26	Executar os casos de teste no sistema alvo	63:00	21:30	0:19	0
17,20,24	Adaptar o sistema alvo	73:30	42:38	0:18	2
27	Elaborar o manual do usuário do sistema	17:00	3:00	0:02	0
28	Converter a base de dados do sistema legado	11:00	3:20	2:00	0
29	Testar o sistema alvo	20:00	12:20	0:02	0
não realizada	Treinar os usuários finais	–	0:00	0:00	0
	Marcos de referência		1:00		
	Total	341:00	**676:29**	**8:37**	**11**

gasto nessa atividade, a aluna gastou três horas para elaborar o diagrama de casos de uso. Assim, o tempo total gasto com atividades de VV&T neste estudo de caso foi de 582:50 horas, ou seja, 549 horas na atividade "Desenvolver o diagrama de casos de uso e elaborar os casos de teste" para elaborar os casos de teste, 21:30 horas na atividade "Executar os casos de teste no sistema alvo" e 12:20 horas na atividade "Testar o sistema alvo". A partir desses valores, observou-se que em torno de 86% de todo o tempo da reengenharia foi gasto principalmente para elaborar os artefatos relacionados a atividades de VV&T.

Como a regra do negócio RN2 foi solicitada pelo usuário após a instanciação do framework (atividade "Criar o sistema alvo no paradigma orientado a objetos") e como

Tabela 3.9: Outros dados coletados durante o estudo de caso

Dado coletado	**Valor**
Quantidade total de requisitos do sistema legado	41 (dos quais 24 foram fornecidos diretamente pelo framework e 3 deles tiveram que ser adaptados, pois não eram completamente similares aos do legado.)
Quantidade de requisitos do sistema legado não encontrados na primeira versão do sistema alvo	17 (sendo 13 relatórios e 4 cadastros).
Quantidade total de linhas de código da primeira versão do sistema alvo	1952 LOC
Quantidade total de linhas de código da versão final do sistema alvo	3362 LOC
Porcentagem de linhas de código herdadas do framework	58%
Quantidade de regras do negócio do sistema legado identificadas pelos casos de teste	0
Quantidade de regras do negócio do sistema legado não identificadas pelos casos de teste	0

a linguagem de padrões GRN possui uma variante do padrão 4 ("Locar o Recurso") que trata da cobrança de multa na locação de recursos, a implementação dessa regra do negócio poderia ser, teoricamente, obtida a partir de uma nova instanciação do framework GREN. Mas, como apresentado na Tabela 3.8, a atividade "Criar o sistema alvo no paradigma orientado a objetos" não foi executada novamente, pois a ferramenta de instanciação GREN-Wizard não apoiava a reengenharia/desenvolvimento iterativo, ou seja, caso o framework fosse instanciado novamente para um determinado sistema, todas as adaptações realizadas manualmente no código fonte das versões anteriores seriam perdidas. Portanto, a implementação da regra RN2 foi feita manualmente na atividade "Adaptar o sistema alvo", iteração 20.

Esses fatos motivaram a criação da ferramenta *GREN-WizardVersionControl* (Cagnin et al., 2004b,e) para controlar as versões das aplicações criadas pelo framework GREN. Essa ferramenta armazena em uma base de dados todas as modificações manuais feitas no código fonte de um determinado sistema, gerado pela instanciação do framework e, durante a geração da nova versão do sistema, as modificações realizadas na versão anterior são detectadas e incorporadas ao código fonte do sistema. Os aspectos de automatização da ferramenta *GREN-WizardVersionControl* são apresentados no Capítulo 6 desta tese.

Como o estudo de caso é um estudo observacional (Wholin et al., 2000), não foi possível obter dados conclusivos com respeito às hipóteses formuladas. No entanto, os resultados obtidos, motivam a realização de experimentos controlados para rejeitar ou não estatisticamente as hipóteses nulas. Observou-se que no estudo de caso conduzido, a maioria dos requisitos do sistema legado estava presente na primeira versão do sistema alvo criada a partir da instanciação do framework GREN. O reúso de linhas de código fonte do framework no sistema alvo foi de 58% (1952 LOC do total de 3362 LOC), não sendo considerado alto de acordo com Yassin e Fayad (2000). No entanto, as demais linhas de código fonte do sistema

alvo que não foram reutilizadas (42%), foram criadas a partir de exemplos de código fonte obtidos da hierarquia de classes do framework. Não foi possível observar o apoio de casos de teste na identificação de regras do negócio do sistema legado durante a reengenharia pois, de acordo com a confirmação do usuário do sistema, o sistema legado realmente não possui implementação de regras do negócio embutida no código fonte.

Discussão

Além da análise quantitativa dos resultados, apresentada nas Tabelas 3.8 e 3.9, uma análise qualitativa foi realizada com base no questionário de *feedback* do estudo de caso, respondido pela aluna de IC que realizou o estudo de caso. Segundo ela, houve dificuldades para converter o banco de dados do sistema legado para o do sistema alvo, pois a especificação dos atributos herdados do framework não pôde ser alterada (ou seja, nome, tipo e tamanho). A justificativa para o tempo excessivo na elaboração dos casos de teste, mesmo utilizando os critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite, foi para verificar se o número de casos de teste criados era suficiente para testar todas as possibilidades de execução dos requisitos do sistema legado. Algumas propriedades fixas do framework, ou seja, os *frozen spots*, não puderam ser modificadas e, assim, o usuário teve que aceitar algumas características impostas pelo framework GREN. A aluna de IC relatou também a impossibilidade de retornar à atividade "Criar o sistema alvo no paradigma orientado a objetos", depois de iniciar a atividade "Adaptar o sistema alvo", sem perder as linhas de código adicionadas manualmente. Relatou também, que a utilização do framework e da documentação do processo facilitou o projeto de reengenharia e que a demora para elaborar e documentar os casos de testes foi o ponto mais negativo da aplicação do PARFAIT.

3.6 Necessidades Evidenciadas a partir dos Estudos de Caso

A partir dos estudos de caso discutidos na Seção 3.4, observou-se a necessidade de evoluir o GREN incorporando a ele alguns requisitos importantes e úteis para o domínio de Gestão de Recursos de Negócios, ou seja, tipo enumerado, tipo multivalorado e emissão de etiquetas. Para isso, um processo de evolução de frameworks de aplicação foi definido e está apresentado no Capítulo 4. Esse processo foi associado ao ARA para garantir que o framework disponível para ser utilizado na reengenharia possa apoiar mais efetivamente a reengenharia de um número maior de sistemas legados que pertencem ao seu domínio.

A partir dos resultados qualitativos e quantitativos observados durante o estudo de caso da Seção 3.5, notou-se a necessidade de uma abordagem de reúso de teste que pudesse apoiar o engenheiro de software na associação de requisitos e casos de teste funcional aos padrões

da linguagem de padrões de análise GRN e, principalmente, permitir o reúso desses a fim de diminuir o tempo e o custo envolvidos nas atividades relacionadas a VV&T do ARA. Essa abordagem foi generalizada para ser aplicada a qualquer linguagem de padrões de análise no domínio de Sistemas de Informação e está apresentada no Capítulo 5. Essa abordagem é outro recurso fornecido pelo ARA, para aumentar sua efetividade. Além disso, observou-se a necessidade de criar uma ferramenta de controle de versão dos sistemas criados a partir do framework GREN, para permitir instanciar o framework novamente para um mesmo sistema sem perder as alterações realizadas no seu código fonte em versões anteriores. Com isso, é possível que o GREN apóie a abordagem iterativa do ARA. Os aspectos funcionais, arquiteturais e de implementação dessa ferramenta são apresentados no Capítulo 6.

3.7 Considerações Finais

Neste capítulo apresentou-se uma visão geral do arcabouço de reengenharia ágil, proposto nesta tese, destacando os recursos necessários para a sua aplicabilidade, dentre eles, o processo de reengenharia, denominado PARFAIT, também apresentado neste capítulo.

Como apresentado, este processo tem como principal seguir as idéias do arcabouço definido, ou seja, fornecer uma versão do sistema alvo o mais rápido possível, com o apoio computacional de frameworks baseados em linguagem de padrões que pertencem ao mesmo domínio do sistema legado, e evoluir essa versão durante diversas iterações até obter o sistema completo. O planejamento do projeto da reengenharia é refeito a cada iteração do processo e avaliam-se os riscos de continuar ou não o projeto de reengenharia. Além disso, PARFAIT conta com a participação dos usuários do sistema legado durante todo o projeto de reengenharia, que validam e refinam os artefatos produzidos. Outro fator importante desse processo é que atividades de VV&T são realizadas durante toda a reengenharia, auxiliando o entendimento do sistema legado, a elicitação e refinamento de requisitos e a validação do produto final.

PARFAIT utiliza padrões para apoiar o entendimento do sistema legado no nível funcional e não arquitetural. A nova arquitetura depende da arquitetura fornecida pelo framework sendo utilizado na reengenharia. Como, em geral, padrões de projeto são utilizados na implementação de frameworks, o sistema alvo, gerado a partir dele, possui as vantagens oferecidas por esses padrões como: manutenibibilidade, flexibilidade, facilidade de entendimento, entre outras.

Além disso, PARFAIT é considerado ágil pois observou-se que incorpora, de alguma forma, a maioria das práticas de XP e também as características ágeis definidas por Abrahamsson et al. (2002).

Neste capítulo apresentaram-se também os três estudos de caso que permitiram observar a viabilidade do processo e a avaliação, adequação e refinamento de suas atividades. Foi

possível também notar e evidenciar a necessidade de recursos mais apropriados, que poderiam aprimorar a aplicação e facilitar o uso do processo e, conseqüentemente, do arcabouço de reengenharia definido no início deste capítulo.

PARFAIT foi utilizado apenas com sistemas de pequeno porte, portanto a sua escalabilidade para sistemas de maior porte não está consolidada. Estudos de caso específicos para observar esses aspectos devem ser conduzidos.

Em síntese, pode-se observar que o arcabouço proposto, por meio do processo PARFAIT, atende grande parte das necessidades de processos e métodos que apóiam a reengenharia de software, pois pretende minimizar principalmente custos e esforços despendidos e alguns dos riscos associados à reengenharia, discutidos por Rosenberg (1996).

É importante esclarecer que os estudos conduzidos neste capítulo e nos dois capítulos subseqüentes são limitados às pessoas envolvidas, ao ambiente onde foram conduzidos e às condições impostas (por exemplo, tamanho do sistema legado, restrição de tempo dos participantes dos estudos, uso de apenas um framework, uso de apenas uma linguagem de padrões de análise). Por exemplo, com relação aos estudos de caso de reengenharia que foram aplicados em sistemas de pequeno porte, ou seja, com menos de 100 mil linhas de código (Sommerville, 2000), o processo PARFAIT comportou-se bem, mas poderia não ser adequado para sistemas de médio porte, ou seja, com até 500 mil linhas de código (Sommerville, 2000)) e também para sistemas de grande porte e/ou em ambiente industrial.

No próximo capítulo apresenta-se o processo que trata da evolução de frameworks de aplicação. Esse processo foi definido e refinado a partir de algumas evoluções do framework GREN. A necessidade de algumas dessas evoluções foram observadas durante a reengenharia com o apoio do PARFAIT, como discutido neste capítulo. É importante ressaltar que esse processo está disponível no arcabouço ARA, apresentado neste capítulo.

Capítulo

4

PREF: Um Processo de Evolução de Frameworks de Aplicação

4.1 Considerações Iniciais

Neste capítulo apresenta-se o Processo de Evolução de Frameworks de Aplicação (PREF), que é proposto nesta tese e foi associado ao arcabouço de reengenharia ágil ARA, proposto no capítulo anterior. Esse processo possui diversas atividades similares às de processos de evolução de software convencional. No entanto, a sua diferença está no controle sistemático da variabilidade de frameworks de aplicação. Variabilidade nesse contexto refere-se à habilidade do framework ser instanciado no domínio de Sistemas de Informação e ser evoluído devido a mudanças no negócio e no mercado, avanços na tecnologia, entre outros. Tal controle de variabilidade objetiva gerenciar a inclusão e alteração de *hot spots* no framework, aumentando o seu potencial de criar um número maior de sistemas a partir de sua instanciação.

De acordo com Sinnema et al. (2004a), tornar a variabilidade explícita em famílias de produtos é um aspecto importante de gerenciamento de variabilidade ((Clauss, 2001; Deelstra et al., 2005) **apud** (Sinnema et al., 2004a)). Em geral, em frameworks, esse gerenciamento pode ser alcançado pelo *cookbook*. Em frameworks baseados em linguagem de padrões, como é o caso do framework GREN, esse gerenciamento é alcançado, em nível mais alto de abstração, pelas variantes dos padrões da linguagem de padrões. No *cookbook* desse tipo de framework a rastreabilidade da variabilidade é explicitada, ou seja, dada uma variante de um determinado

padrão é possível saber quais classes do framework devem ser instanciadas e quais métodos devem ser sobrepostos. PREF não se preocupa com a modelagem de variabilidade, apenas com a atualização da documentação do framework que a possui.

O processo de evolução de frameworks foi definido e refinado a partir de algumas evoluções feitas no framework GREN, tanto relacionadas a requisitos funcionais quanto a requisitos não funcionais. As evoluções relacionadas a requisitos funcionais foram motivadas após o uso do GREN na reengenharia de software, com o apoio do PARFAIT e, conseqüentemente, do arcabouço ARA, conforme mencionado no Capítulo 3, e foram conduzidas pela autora desta tese. A necessidade da evolução relacionada a um requisito não funcional foi motivada desde a concepção do GREN, no entanto foi somente conduzida posteriormente em um trabalho de mestrado (Silva, 2004b).

Este capítulo está organizado da seguinte forma: na Seção 4.2 apresentam-se as atividades do processo proposto, destacando-se principalmente aquelas relacionadas ao controle de variabilidade do framework. Na Seção 4.3 apresentam-se duas evoluções realizadas no framework GREN com o apoio do processo proposto neste capítulo. Apesar de não ter sido coletado dado quantitativo das evoluções conduzidas, observou-se que esse processo é efetivo no contexto em que foi aplicado após análise qualitativa dos resultados obtidos. Na Seção 4.4 são apresentadas as conclusões parciais do trabalho, enfocando o processo definido neste capítulo.

4.2 Processo PREF

O processo PREF (Cagnin et al., 2004d,f, 2005b) contém oito atividades que são executadas de maneira incremental e iterativa, permitindo ao mantenedor retornar a qualquer atividade anteriormente executada, a fim de refinar os artefatos produzidos, conforme apresentado na Figura 4.1. O número das atividades, apresentado nessa figura, não representa a seqüência de execução das atividades e é utilizado para facilitar a descrição do processo. As atividades 1 e 3 são específicas para controlar a variabilidade de frameworks e ajudam identificar os *hot spots* que devem ser neles incorporados. As atividades 2, 4, 5, 6, 7 e 8 são similares a atividades de processos de manutenção de software convencional (Pfleeger e Bohner, 1999; Polo et al., 1999; Rajlich, 2001; Yau e Collofello, 1980). Dentre essas, as atividades 2, 4, 6 e 7 consideram, no mínimo, um dos três passos básicos de processos de manutenção, de acordo com Boehm (1976), ou seja, entender, modificar e re-validar o software.

No PREF, o mantenedor possui dois papéis distintos: engenheiro de aplicação e engenheiro de domínio. O engenheiro de aplicação realiza manutenção diretamente no sistema criado a partir da instanciação do framework. Esse tipo de manutenção é feito quando os requisitos do sistema, não cobertos pelo framework, não devem ser nele incorporados porque: 1) o requisito é específico do sistema e não pertence ao domínio do framework, ou 2)

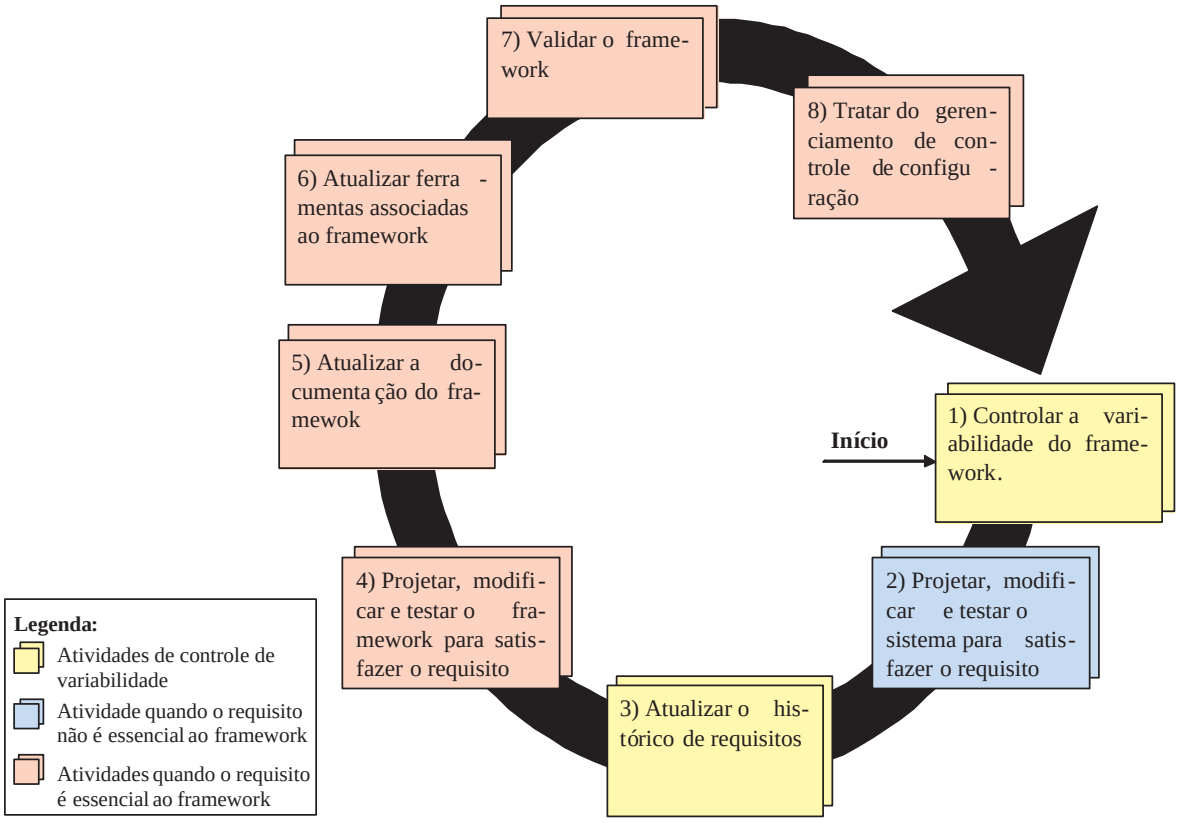


Figura 4.1: Visão Geral do PREF

o requisito pertence ao domínio do framework, mas não há viabilidade técnica ou financeira para incorporá-lo ao framework.

O engenheiro de domínio realiza manutenção no framework quando o requisito pertence ao domínio do framework e é considerado relevante para ser nele incorporado, a fim de permitir que um número maior de sistemas seja criado a partir de sua instanciação. Nesse caso, é necessário tratar do Gerenciamento de Controle de Configuração do framework, das ferramentas associadas e dos sistemas criados por meio da instanciação do framework. Isso é tratado pelo PREF na atividade 8.

Salienta-se que diversas equipes geograficamente distantes e trabalhando em paralelo podem também utilizar o PREF. Nesse caso, um engenheiro de domínio (ou um grupo deles) que pertence à equipe de evolução de framework pode assumir a função de controlador dos pedidos de manutenção. Esses pedidos são encaminhados pelos engenheiros de aplicação, por outros engenheiros de domínio ou pelos próprios usuários dos sistemas criados a partir da instanciação do framework. Tal controlador tem a responsabilidade de receber os pedidos de manutenção, analisar se os requisitos não cobertos pelo framework são essenciais ou não ao seu domínio, determinar o nível de prioridade de cada um desses pedidos, agrupá-los de acordo com funções em comum e, então, distribuí-los para equipes apropriadas para executar cada tipo de manutenção.

Para que o uso do processo seja bem sucedido, é necessário que o mantenedor tenha conhecimento prévio do domínio, da funcionalidade e da arquitetura do framework. O PREF foi utilizado, até o momento, somente para realizar atividades de manutenção **perfectiva** (modifica o código fonte para substituir, adicionar ou estender funcionalidade) (Chapin et al., 2001). No entanto, acredita-se que atividades de manutenção dos tipos **corretiva** (modifica o código fonte para consertar defeitos de funcionalidade), **adaptativa** (modifica o código fonte para acomodar mudanças em seu ambiente externo (ou seja, CPU, SO, etc) ou recursos utilizados, sem alterar sua funcionalidade), **preventiva** (modifica o código fonte para evitar ou reduzir atividades de manutenção futuras, sem alterar a funcionalidade e a tecnologia existentes) e de **desempenho** (modifica o código fonte para alterar as características ou propriedades de desempenho do software) (Chapin et al., 2001) também podem ser conduzidas aplicando as atividades 4 a 8 do processo, após estudos de viabilidade técnica, financeira, entre outras. Atividades de manutenção **redutivas** (modifica o software para restringir ou reduzir funcionalidade) (Chapin et al., 2001) podem também ser conduzidas seguindo as atividades de 4 a 8, mas alterando apropriadamente a redação da atividade 4 por "Projetar a redução, reduzir e testar as funções remanescentes no framework".

As atividades 2 e 3 do PREF são conduzidas pelo engenheiro de aplicação quando se decide implementar o requisito não coberto pelo framework apenas no sistema gerado por ele, enquanto que as atividades 3, 4, 5, 6, 7 e 8 são conduzidas pelo engenheiro de domínio quando se decide implementar os requisitos não cobertos pelo framework nele próprio, para que outros sistemas com o mesmo requisito possam ser gerados por meio da sua instanciação. A decisão do local de implementação do requisito não coberto pelo framework é feita na atividade 1 pelo engenheiro de domínio.

Ressalta-se que o processo é iniciado quando, durante a instanciação, um requisito ou conjunto de requisitos (funcional ou não-funcional) não é coberto pelo framework, ou seja, somente com a instanciação do framework não é possível construir o sistema com todos os seus requisitos, existindo pelo menos um requisito do sistema não fornecido pelo framework. No caso de frameworks cuja construção é baseada em linguagem de padrões, isso é observado anteriormente à sua instanciação, por exemplo, durante a aplicação do PARFAIT na atividade "Desenvolver o diagrama de classes do sistema alvo". Em outros frameworks, a não cobertura do framework em relação a alguns requisitos pode ser observada por meio do seu *cookbook*. Nesse caso, isso não é feito de modo tão direto e rápido, pois o *cookbook* em geral contém informações relacionadas à camada de projeto e de implementação do framework e não à de análise. Em geral, o processo PREF é também aplicado periodicamente pelo engenheiro de domínio para decidir se o framework deve ou não ser evoluído, baseando-se em um Histórico de Requisitos [1]. Isso para que o tempo de evolução do framework não interfira

[1]Contém uma lista de requisitos não cobertos previamente pelo framework. Essa lista apóia o engenheiro de domínio decidir se um requisito deve ser incorporado ou não ao framework. Caso o requisito seja

no tempo dos projetos de desenvolvimento e de reengenharia, que utilizam o framework como apoio computacional.

A seguir, estão descritas cada uma das atividades do processo PREF seguindo um formato que contém alguns elementos fundamentais do arcabouço do RUP (Kruchten, 2000): atividades, passos, papéis, artefatos de entrada e de saída, gabaritos e guia de ferramenta de apoio computacional.

Atividade 1) Controlar a variabilidade do framework

Objetivo: Nesta atividade o engenheiro de domínio deve tomar a mais importante decisão quando se depara com um pedido de manutenção no framework ou quando aplica o processo periodicamente, ou seja, se um requisito (ou um conjunto de requisitos) não coberto pelo framework:

- é considerado essencial [2] ao domínio do framework: há probabilidade significativa [3] de estar presente em outros sistemas, portanto pode ser incorporado ou não ao framework. O requisito não é incorporado ao framework quando não há viabilidade técnica (por exemplo, não há engenheiro de domínio com tempo disponível para realizar a evolução do framework), financeira (relacionada, por exemplo, com o custo de generalizar a implementação do requisito para ser incorporado ao framework), de escopo (por exemplo, o requisito não pertence ao escopo definido para o framework), entre outras.

- não é considerado essencial ao domínio do framework: portanto o requisito não deve ser incorporado ao framework, mas apenas ao sistema gerado a partir de sua instanciação.

O requisito pode ser implementado primeiramente no sistema gerado pelo framework e, posteriormente, no próprio framework mesmo quando é, a priori, considerado essencial. Isso é feito para facilitar a implementação do requisito no framework posteriormente, pois a complexidade de entendimento e o grau de dificuldade de implementação são menores no sistema do que no framework.

Papel: Engenheiro de domínio.

Artefatos de Entrada: Requisito não coberto pelo framework, Histórico de Requisitos.

Apoio Computacional: Editor de texto.

Gabarito: Na Tabela 4.1 apresenta-se o gabarito da análise dos requisitos não cobertos pelo framework, que deve conter uma listagem das situações em que o requisito (ou conjunto de requisitos) sendo analisado pode ocorrer novamente em outras situações no domínio do framework. Veja também o item *Gabarito* da atividade "Atualizar o Histórico de Requisitos".

incorporado ao framework, a lista é atualizada com a versão do framework em que o requisito foi implementado e a `situação` do requisito muda de "pendente" para "atendido".

[2] Pertence ao domínio do framework e sua ausência pode limitar a instanciação do framework para um número significativo de sistemas.

[3] Essa probabilidade é subjetiva e deve ser determinada pelo engenheiro de domínio, de acordo com sua experiência no domínio do framework.

Tabela 4.1: Análise dos requisitos não cobertos pelo framework

ID Requisito	Situações em que ele pode ocorrer novamente
...	...

Artefatos de Saída: Análise dos requisitos não cobertos pelo framework.

Passo:

1. **Analisar se o requisito (ou conjunto de requisitos) é essencial ao domínio do framework:** Para decidir se o requisito (ou conjunto de requisitos) não coberto pelo framework é essencial ou não ao seu domínio, é necessário que o engenheiro de domínio utilize as seguintes diretrizes:

   1) utilize o conhecimento no domínio;

   2) construa uma lista contendo possíveis e prováveis situações em que o requisito pode ocorrer novamente, de acordo com o gabarito apresentado na Tabela 4.1. Caso existam no mínimo duas situações em que o requisito pode ocorrer novamente, há indícios de que ele pode ser essencial ao domínio do framework.

   3) consulte o Histórico de Requisitos para verificar, de acordo com Roberts e Johnson (1998), se há pelo menos mais duas ocorrências do requisito que está sendo analisado, cuja `situação` seja igual a "pendente". Caso positivo, há forte indício de que se trata de um requisito essencial ao domínio do framework e que deve ser nele incorporado. Deve-se notar que requisitos aparentemente diferentes podem estar relacionados a uma única solução de manutenção.

Se o requisito for considerado como essencial ao domínio e se for decidido incorporá-lo ao framework, executar as atividades 4 a 8 e 3, senão executar as atividades 2 e 3.

Atividade 2) Projetar, modificar e testar o sistema para satisfazer o requisito (ou conjunto de requisitos)

Objetivo: Uma vez decidido que o requisito não deve ser incorporado ao framework, ele é implementado, nesta atividade, diretamente no sistema gerado a partir da instanciação do framework. Para realizar a modificação no sistema, é necessário estudar as suas classes e métodos, bem como as superclasses herdadas do framework para que as modificações realizadas não danifiquem o comportamento esperado pelo sistema.

Papéis: Engenheiro de aplicação.

Artefatos de Entrada: Código fonte do sistema gerado a partir do framework, superclasses herdadas do framework, documentação do sistema (caso exista).

Apoio Computacional: Ferramenta de desenvolvimento.

Artefatos de Saída: Código fonte do sistema alterado, Documentação do sistema alterada.

Passos:

1. **Verificar se a versão atual do sistema foi submetida ao gerenciamento de controle de configuração:** Antes de modificar o sistema é necessário verificar se sua versão atual foi submetida ao Gerenciamento de Controle de Configuração. Caso não tenha sido, o passo "Tratar do gerenciamento de controle de configuração do sistema" da atividade "Tratar do gerenciamento de controle de configuração" deve ser executado nesse momento.

2. **Projetar as modificações no sistema para satisfazer o requisito (ou conjunto de requisitos):** Neste passo, é necessário estudar as classes do sistema e as subclasses herdadas do framework com o apoio da documentação existente do sistema. Isso é feito para que a solução de projeto adotada não interfira no comportamento fornecido pelas superclasses do framework. O projeto deve ser feito visando a flexibilidade de alterações porque, se posteriormente for tomada a decisão de incorporar o requisito no framework, o trecho do código inserido/alterado no sistema será provavelmente usado como base para generalizar a implementação que será feita no framework. A documentação existente do sistema deve ser alterada para refletir o projeto das modificações projetadas. Este passo corresponde ao passo básico "entender" de processos de manutenção de software (Boehm, 1976).

3. **Modificar o sistema:** As classes, atributos e métodos são adicionados e/ou alterados no sistema, de acordo com o projeto do passo anterior. Este passo corresponde ao passo básico "modificar" de processos de manutenção de software (Boehm, 1976).

4. **Testar o sistema:** Depois de modificar o sistema para satisfazer o requisito não coberto pelo framework, deve-se testá-lo, primeiramente com teste de unidade nas classes afetadas e, em seguida, aplicando testes de integração e de sistema. Em todos esses casos, se possível, utilizar casos de teste de regressão, identificando casos de teste associados com as partes do sistema que foram modificadas, para verificar especialmente se nenhuma parte do sistema foi danificada com a implementação realizada. Isso pode ser apoiado por técnicas, critérios e ferramentas de teste OO existentes (Binder, 1999; Pressman, 2005). Recomenda-se documentar todos os casos de teste utilizados neste passo para que possam ser utilizados no passo "Testar o framework" da atividade "Projetar, modificar e testar o framework para satisfazer o requisito". Este passo corresponde ao passo básico "re-validar" de processos de manutenção de software (Boehm, 1976).

Atividade 3) Atualizar o histórico de requisitos

Objetivo: Nesta atividade o Histórico de Requisitos é atualizado, de acordo com as seguintes situações:

- quando o requisito (ou conjunto de requisitos) não é essencial ao framework: o Histórico de Requisitos é atualizado armazenando diversas informações, conforme apresentadas na Tabela 4.2. A `situação` do requisito é atribuída inicialmente como "pendente".
- quando o requisito (ou conjunto de requisitos) é essencial ao framework:
  - ao iniciar a implementação do requisito no framework, a sua `situação` no Histórico de Requisitos é alterada para "sendo atendido". Isso é feito para que outros engenheiros de domínio saibam que tal requisito não é coberto pelo framework no momento, mas que isso está sendo providenciado.
  - após o término da implementação do requisito no framework, a sua `situação` no Histórico de Requisitos é alterada para "atendido".
  - durante o Gerenciamento de Controle de Configuração, a `versão` do framework em que o requisito é coberto, é atualizada no Histórico de Requisitos. Isso é feito para que o engenheiro de domínio saiba em qual versão do framework tal requisito é fornecido.

Papéis: Engenheiro de Domínio e Engenheiro de Aplicação.

Artefatos de Entrada: Histórico de Requisitos.

Apoio Computacional: Editor de Texto.

Gabarito: Na Tabela 4.2 apresenta-se o gabarito do Histórico de Requisitos que é atualizado nesta atividade. Esse histórico contém as seguintes informações: o `identificador` e a `descrição` de cada requisito, a `solução` de projeto adotada e implementada no sistema, a `razão` do "porque" o requisito não é essencial ao domínio do framework, o nome do `sistema` em que o requisito foi implementado e `quando` a manutenção foi realizada (na reengenharia ou no desenvolvimento de software), o `tipo de manutenção` realizada, a `situação` do requisito em relação ao framework ("pendente" - não incorporado ao framework, "sendo atendido" - sendo implementado no framework e "atendido" - incorporado ao framework) e a `versão` do framework em que o requisito foi incorporado, se for o caso.

Tabela 4.2: Gabarito do Histórico de Requisitos

ID	Descrição do Requisito	Solução de projeto implementada no sistema	Razão do "porque" o requisito não é essencial	Sistema	Tipo de Manutenção	Situação	Versão do framework
...	...	...	...	...	...	...	...

Artefatos de Saída: Histórico de Requisitos atualizado.

Passos: O passo 1 é executado pelo engenheiro de aplicação, enquanto os passos 2 e 3 são executados pelo engenheiro de domínio.

1. **Registrar o requisito (ou conjunto de requisitos) no histórico de requisitos:** Depois de modificar o sistema para atender a um determinado requisito (ou conjunto de requisitos), o engenheiro de aplicação deve informar sobre a solução adotada. Isso deve ser feito armazenando no Histórico de Requisitos (Tabela 4.2) as seguintes informações: o `identificador` e a `descrição` de cada requisito, a `solução` de projeto adotada, a `razão` do "porque" o requisito não é essencial ao domínio do framework, o nome do `sistema` em que o requisito foi implementado, `quando` a manutenção foi realizada e a `situação` do requisito igual a "pendente", pois o requisito não foi incorporado ao framework.

   Como a quantidade de requisitos no Histórico de Requisitos pode aumentar significativamente, sugere-se que o engenheiro de aplicação armazene o requisito (ou conjunto de requisitos) em uma linha do histórico, logo abaixo daquelas que contém requisitos similares (caso existam) ao que está sendo armazenado. Isso para facilitar a busca de ocorrências de requisitos no Histórico de Requisitos durante a execução da atividade "Controlar a variabilidade do framework". Caso não exista requisito similar no Histórico de Requisitos, sugere-se que o engenheiro de aplicação armazene o requisito no final do histórico.

2. **Atualizar a situação do requisito (ou conjunto de requisitos) no histórico de requisitos.** Este passo deve ser realizado quando:

   (a) decide-se por incorporar o requisito (ou conjunto de requisitos) ao framework. Nesse caso, a `situação` do requisito deve ser atualizada como "sendo atendido";

   (b) o requisito (ou conjunto de requisitos) é incorporado ao framework. Nesse caso, a `situação` do requisito deve ser atualizada como "atendido".

3. **Atualizar a versão do framework em que o requisito (ou conjunto de requisitos) foi incorporado.** Neste passo, o item `versão` do framework do Histórico de Requisitos é atualizado com o número da versão do framework em que o requisito (ou conjunto de requisitos) foi incorporado. Portanto, somente deve ser executado depois que o framework foi submetido ao Gerenciamento de Controle de Configuração (passo "Tratar do gerenciamento de controle de configuração do framework/ferramenta de instanciação" da atividade "Tratar do gerenciamento de controle de configuração").

Atividade 4) Projetar, modificar e testar o framework para satisfazer o requisito (ou conjunto de requisitos)

Objetivo: Nesta atividade, requisitos que são considerados essenciais ao domínio, pela decisão tomada na atividade "Controlar a variabilidade do framework", devem ser implementados no framework antes que o novo sistema seja gerado a partir de sua instanciação, caso exista viabilidade para isso. Ressalta-se que, para executar essa atividade, é necessário estudar antecipadamente ou ter um conhecimento aprofundado da arquitetura do framework e de sua hierarquia de classes.

Papel: Engenheiro de Domínio.

Artefatos de Entrada: Framework, documentação do framework (*cookbook*, diagrama de classes do projeto, documentação da hierarquia de classes do framework, entre outras).

Apoio Computacional: Ferramenta de desenvolvimento do framework.

Artefatos de Saída: Framework atualizado.

Passos:

1. **Verificar se a versão atual do framework foi submetida ao gerenciamento de controle de configuração:** Antes de fazer qualquer alteração no código fonte do framework é necessário verificar se sua versão atual foi submetida ao Gerenciamento de Controle de Configuração. Caso não tenha sido, o passo "Tratar do gerenciamento de controle de configuração do framework/ferramenta de instanciação" da atividade "Tratar do gerenciamento de controle de configuração" deve ser executado nesse momento.

2. **Projetar as modificações no framework para satisfazer o requisito (ou conjunto de requisitos):** Primeiramente, deve-se decidir como o requisito pode ser incorporado ao framework do ponto de vista de projeto, observando os *hot spots* existentes e as dependências entre eles. As soluções de projeto dos requisitos que já foram implementadas em sistemas gerados a partir da instanciação do framework e registradas no Histórico de Requisitos, devem servir como base para a evolução do framework e, se possível, devem ser generalizadas a partir do conhecimento que o engenheiro de domínio possui em relação a arquitetura, hierarquia de classes e domínio do framework. O diagrama de classes do projeto do framework deve ser analisado, observando quais classes, relacionamentos, métodos, e atributos devem ser adicionados e/ou modificados para permitir a implementação do requisito (ou conjunto de requisitos). Para isso, é necessário executar o passo "Atualizar a documentação de análise e projeto" da atividade "Atualizar a documentação do framework". Outro aspecto importante é realizar análise de impacto, observando os efeitos dessas modificações na hierarquia de classes do framework, a fim de projetar corretamente as modificações. Além disso, é necessário levar em consideração impactos

potenciais nos sistemas gerados, na documentação e também nas pessoas, ou seja, mantenedores do framework (engenheiros de domínio), usuários diretos (engenheiros de aplicação) e usuários indiretos (usuários dos sistemas gerados a partir da instanciação do framework). O Histórico de Requisitos deve ser atualizado com `situação` do requisito igual a "sendo atendido" (passo "Atualizar a situação do requisito" da atividade "Atualizar o histórico de requisitos"). Este passo corresponde ao passo básico "entender" de processos de manutenção de software (Boehm, 1976).

3. **Modificar o framework:** Neste passo, o framework é modificado de acordo com o projeto realizado no passo anterior. Classes, atributos e métodos podem ser adicionados e/ou alterados. No caso de adição de novas classes na hierarquia de classes do framework, é importante distinguir quais classes farão parte da estrutura fixa do framework e quais se tornarão mais flexíveis, aplicando-se, por exemplo, meta-padrões (Pree, 1999) e padrões de projeto (Gamma et al., 1995) que, portanto, farão parte da estrutura variável do framework. Refatoração (Fowler et al., 1999) pode ser realizada nesse momento para melhorar a legibilidade do código e para otimizar algoritmos existentes, se for o caso. Após cada modificação no framework é necessário executar o passo "Atualizar o *cookbook*" da atividade "Atualizar a documentação do framework". Este passo corresponde ao passo básico "modificar" de processos de manutenção de software (Boehm, 1976).

4. **Testar o framework:** O engenheiro de domínio deve testar as funções implementadas, utilizando algum critério de teste orientado a objetos (Binder, 1999) ou alguma técnica específica para teste de framework (Dallal e Sorenson, 2002; Jeon et al., 2002), a fim de garantir que todas as partes do framework que refletem as modificações feitas foram devidamente testadas. Se os métodos/classes implementados no framework já tiverem sido previamente implementados durante modificações nos sistemas gerados a partir da instanciação do framework (passo "Testar o sistema" da atividade "Projetar, modificar e testar o sistema para satisfazer o requisito"), pode-se reutilizar os testes previamente criados e adaptá-los para as novas necessidades. Além disso, é necessário retomar os testes utilizados nas evoluções anteriores do framework e, se for o caso, atualizá-los para se adequarem à modificação conduzida. Recomenda-se documentar todos os casos de teste utilizados neste passo. O Histórico de Requisitos deve ser atualizado com `situação` do requisito igual a "atendido" (passo "Atualizar a situação do requisito" da atividade "Atualizar o histórico de requisitos"). Este passo corresponde ao passo básico "re-validar" de processos de manutenção de software (Boehm, 1976).

Atividade 5) Atualizar a documentação do framework

Objetivo: Nesta atividade toda a documentação do framework deve ser atualizada para refletir as modificações feitas na atividade "Projetar, modificar e testar o framework para satisfazer o requisito".

Papel: Engenheiro de domínio.

Artefatos de Entrada: Documentação do framework, *cookbook*.

Apoio Computacional: Editor de texto, ferramentas de apoio à edição de diagramas.

Artefatos de Saída: Documentação do framework atualizada, *cookbook* atualizado.

Passos:

1. **Atualizar a documentação de análise e projeto:** Toda a documentação de análise e de projeto do framework (diagrama de classes de análise e projeto, documentação da hierarquia de classes, diagrama de casos de uso, entre outras) deve ser atualizada para refletir as modificações realizadas no framework. Se a construção do framework é baseada em linguagem de padrões, é necessário verificar se o requisito incorporado ao framework representa um padrão ou uma variante de um padrão. Caso positivo, a documentação da linguagem de padrões deve ser também atualizada.

2. **Atualizar o *cookbook*:** Neste passo, o *cookbook* do framework é atualizado para refletir as alterações ocorridas nos passos da instanciação do framework devido a modificações no framework, a fim de permitir que a instanciação caixa-branca seja realizada corretamente.

Atividade 6) Atualizar ferramentas associadas ao framework (por exemplo, ferramenta de instanciação)

Objetivo: Se o framework possui ferramentas associadas, elas devem ser atualizadas apropriadamente nesta atividade para refletir as mudanças feitas no framework. Como nem todos os frameworks possuem ferramentas associadas, esta atividade é opcional.

Papel: Engenheiro de domínio.

Artefatos de Entrada: Ferramenta de apoio à instanciação do framework, *cookbook*, guia da ferramenta de apoio à instanciação (caso exista).

Apoio Computacional: Ferramenta de desenvolvimento da ferramenta.

Artefato de Saída: Ferramenta de apoio à instanciação do framework atualizada, guia da ferramenta de apoio à instanciação atualizado (caso exista).

Passos:

1. **Verificar se a versão atual da ferramenta foi submetida ao gerenciamento de controle de configuração:** como na evolução do framework, antes de atualizar

as ferramentas associadas a ele é necessário verificar se elas já foram submetidas ao Gerenciamento de Controle de Configuração. Caso não tenham sido, o passo "Tratar do gerenciamento de controle de configuração do framework/ferramenta de instanciação" da atividade "Tratar do gerenciamento de controle de configuração" deve ser executado nesse momento.

2. **Projetar, modificar e testar a ferramenta de instanciação do framework para satisfazer o requisito (ou conjunto de requisitos):** Para atualizar as ferramentas associadas ao framework é necessário considerar as mesmas questões necessárias para realizar qualquer tipo de manutenção, como já mencionadas na atividade "Projetar, modificar e testar o framework para satisfazer o requisito", ou seja, entender a arquitetura e a hierarquia de classes da ferramenta, o código fonte, testar o sistema. No caso específico de atualização da ferramenta de instanciação, o *cookbook*, atualizado no passo "Atualizar o *cookbook*" da atividade "Atualizar a documentação do framework", pode servir como um guia para indicar as mudanças que devem ser realizadas na ferramenta, já que ela deve refletir o processo de instanciação atualizado. Para complementar a atividade de teste, o passo "Instanciar o framework caixa-preta" da atividade "Validar o framework" é executado. Neste passo atualiza-se também o guia do usuário da ferramenta de apoio à instanciação (caso exista) para refletir as alterações ocorridas na sua interface e também nos procedimentos de instanciação do framework, a fim de permitir que a instanciação caixa-preta do framework seja realizada corretamente. Caso haja documentação de projeto da ferramenta, por exemplo, diagrama de classes, documentação de sua hierarquia de classes, entre outras; é necessário atualizá-la refletindo as mudanças feitas na ferramenta de apoio à instanciação. Este passo é similar aos passos básicos "entender, modificar e re-validar" de processos de manutenção de software (Boehm, 1976).

Atividade 7) Validar o framework

Objetivo: Ressalta-se que a única forma de validar um framework é por meio de instanciações e, portanto, quanto mais aplicações específicas forem criadas melhor é sua validação ((Bosch et al., 1999) **apud** (Ré, 2002)).

Esta atividade é executada com o apoio de técnicas de regressão para verificar se as funções implementadas satisfazem o requisito pretendido e se as funções existentes não foram danificadas com a implementação das novas. Ressalta-se que a validação do framework é importante também para validar, consistir e corrigir o *cookbook*, atualizado na atividade "Atualizar a documentação do framework", e o guia da ferramenta de apoio à instanciação do framework (caso exista), atualizado na atividade "Atualizar ferramentas associadas ao framework", pois esses são utilizados durante as instanciações caixa-branca e caixa-preta.

Esta atividade corresponde ao passo básico "re-validar" de processos de manutenção de software.

Papéis: Engenheiro de domínio e Engenheiro de aplicação

Artefatos de Entrada: Framework, *cookbook*, guia da ferramenta de apoio à instanciação (caso exista).

Apoio Computacional: Ferramenta de apoio à instanciação do framework (caso exista).

Artefatos de Saída: Framework validado.

Passos: Os passos a seguir devem ser realizados com a cooperação do engenheiro de domínio e do engenheiro de aplicação para verificar se o requisito (ou conjunto de requisitos) foi incorporado corretamente ao framework.

1. **Instanciar o framework caixa-branca:** Todas as partes do framework que refletem as modificações devem ser validadas por meio de diversas instanciações caixa-branca. Para isso, é necessário seguir o processo de instanciação atualizado no *cookbook* (passo "Atualizar o *cookbook*" da atividade "Atualizar a documentação do framework") e considerar nas instanciações os novos requisitos incorporados. Isso pode ser feito tomando como base o Histórico de Requisitos.

2. **Instanciar o framework caixa-preta (caso exista):** Todas as partes do framework que refletem as modificações devem ser validadas por meio de diversas instanciações caixa-preta. Para isso, é necessário seguir o guia da ferramenta de apoio à instanciação atualizado (no passo "Atualizar outros documentos relacionados" da atividade "Atualizar a documentação do framework") e considerar durante as instanciações os novos requisitos incorporados.

Atividade 8) Tratar do gerenciamento de controle de configuração

Objetivo: O controle de configuração de frameworks deve ser realizado com muita precaução, pois evoluções em frameworks podem mudar o seu projeto e, conseqüentemente, o dos sistemas gerados a partir da instanciação deles. Como resultado, esses sistemas podem não aderir ao novo projeto e podem deixar de fornecer o comportamento desejado. Cada alteração feita no framework implica em definir em quais versões do framework a manutenção efetuada será incorporada. Esta atividade é essencialmente similar ao Gerenciamento de Controle de Configuração de outros tipos de software, mas pode se tornar mais complexa devido a inúmeras versões do framework e ferramentas de apoio a ele associadas. Ressalta-se que esta atividade fornece apenas diretrizes de como pode ser feito o Gerenciamento de Controle de Configuração (maiores detalhes são obtidos em (Pressman, 2005; Sommerville, 2000)), ressaltando alguns problemas que podem surgir quando o controle de versões nos frameworks e nos sistemas gerados não é feito adequadamente.

Papel: Engenheiro de domínio.

Artefatos de Entrada: Framework, Ferramenta de apoio à instanciação do framework, Sistema gerado a partir da instanciação do framework.

Apoio Computacional: Ferramenta de controle de versão (por exemplo, CVS (Concurrent Versions System, 2004), *VersionControl* (Soares et al., 2000), Ferramenta de controle de versão dos sistemas gerados a partir do framework (por exemplo, *GREN-WizardVersionControl* descrita no Capítulo 6 desta tese).

Artefato de Saída: Repositório das ferramentas de controle de versão atualizado.

Passos:

1. **Tratar do gerenciamento de controle de configuração do framework/ferramenta de instanciação:** Entre as decisões mais importantes a serem tomadas, por exemplo, durante o Gerenciamento de Controle de Configuração de frameworks estão: Os sistemas gerados a partir da instanciação do framework devem ser modificados para incorporar a evolução do framework? A partir de que momento a manutenção realizada será incorporada à versão operacional do framework? Se o framework possui mais de uma versão operacional (para diferentes plataformas, por exemplo) e ferramentas de instanciação, deve-se decidir se e quando elas incorporarão a nova funcionalidade. O engenheiro de domínio deve decidir qual é a solução mais adequada ao seu contexto e aplicar as técnicas de Gerenciamento de Controle de Configuração de software. Por exemplo, no caso de sistemas gerados e que tenham passado por alterações manuais no código fonte (ou seja, adaptações), deve-se tomar outras precauções ao considerar a nova versão do framework, pois o código incluído no sistema pode entrar em conflito com as novas funções implementadas no framework. Isso pode ser resolvido possivelmente de duas maneiras:

   - criar o sistema novamente: quando o sistema foi gerado com o apoio da ferramenta de instanciação do framework e a sua especificação estiver armazenada, de alguma forma, em um histórico, ele pode ser facilmente gerado novamente utilizando a nova versão da ferramenta de instanciação, e, conseqüentemente, a nova versão do framework. Porém, é necessário refazer as adaptações na nova versão do sistema que foram feitas em versões anteriores. Além disso, podem ocorrer mudanças na estrutura da base de dados, havendo necessidade de migrar os dados da versão anterior do sistema. No caso específico do GREN, com o uso da ferramenta *GREN-WizardVersionControl*, apresentada no Capítulo 6 desta tese, é possível criar uma nova versão do sistema com as adaptações incorporadas e também há a possibilidade de alterar a estrutura da base de dados e manter os dados de versões anteriores.
   - não criar o sistema novamente: o sistema continua operando na versão do framework em que foi criado. Nesse caso, é necessário um controle rigoroso de

versões para gerenciar as versões disponíveis do framework e de seus sistemas correspondentes. Uma solução para isso seria manter um documento em que seja possível rastrear cada versão do framework e da sua ferramenta de instanciação (caso exista), os requisitos do domínio que atendem e os sistemas correspondentes, criados a partir de tal versão.

Para finalizar este passo, o Histórico de Requisitos deve ser atualizado com a versão do framework ao qual o requisito foi incorporado (passo "Atualizar a versão do framework em que o requisito foi incorporado" da atividade "Atualizar o histórico de requisitos").

2. **Tratar do gerenciamento de controle de configuração do sistema:** O controle de versões dos sistemas gerados a partir do framework é, em geral, similar ao controle de versões de software convencional. No caso de sistemas que tenham passado por alterações manuais no código fonte (ou seja, adaptações), deve-se tomar precauções relacionadas ao controle de versões. Isso porque, se houver necessidade de uma nova instanciação do framework para acrescentar algum novo requisito no sistema, o código incluído manualmente no sistema é perdido. No contexto do framework GREN, a ferramenta *GREN-WizardVersionControl* evita esse problema.

4.3 Uso do Processo PREF

O processo PREF foi utilizado para apoiar algumas evoluções do framework GREN relacionadas a requisitos funcionais e não funcionais.

A autora utilizou o PREF para incorporar no framework GREN alguns requisitos funcionais não cobertos por ele. A ausência desses requisitos foi evidenciada durante a condução de alguns estudos de caso de reengenharia utilizando o PARFAIT (Capítulo 3) e, conseqüentemente, o arcabouço ARA. O PREF foi também utilizado em um trabalho de mestrado (Silva, 2004b), para incorporar ao framework GREN um sub-sistema de controle de segurança. A necessidade desse requisito não funcional foi notada desde a construção do framework, mas não havia sido implementado devido à ausência de tempo e à decisão do escopo da primeira versão do framework.

Nesta Seção são relatadas resumidamente as evoluções feitas no framework GREN com o apoio do PREF. Será dado enfoque apenas às atividades que são inerentes à evolução de frameworks.

Atividade 1) Controlar a variabilidade do framework

Na Tabela 4.3 apresenta-se parcialmente o Histórico de Requisitos contendo alguns requisitos não cobertos pelo framework GREN durante a reengenharia, utilizando o PARFAIT, e durante o desenvolvimento de sistemas no domínio de Gestão de Recursos de Negócios.

Portanto, esses requisitos foram implementados nos sistemas gerados a partir da instanciação do framework GREN (atividade "Projetar, modificar e testar o sistema para satisfazer o requisito" do PREF) e possuem situação igual a "pendente". Alguns desses requisitos poderiam ser satisfeitos não completamente pelo framework, por exemplo, os requisitos 1 a 5 poderiam ser implementados como atributos do tipo *string*, mas isso poderia causar problemas de inconsistências e redundâncias no futuro. A necessidade de outros requisitos (por exemplo, de controle de segurança, de suporte a diferentes tipos de bases de dados e a distribuição (Braga, 2003)) foi observada durante o desenvolvimento do próprio framework, mas não foram implementados por falta de tempo e também devido a decisão de escopo.

Para decidir sobre a evolução do framework GREN, realizou-se uma análise dos requisitos listados no Histórico de Requisitos, a fim de verificar quais eram essenciais ao domínio de tal framework. Essa análise foi feita em duas etapas: 1) analisou-se, de acordo com o conhecimento do engenheiro de domínio, em quais outras situações no domínio de Gestão de Recursos de Negócios os requisitos poderiam ocorrer, conforme apresentado na Tabela 4.4; e 2) verificou-se o número de ocorrência de cada requisito (cuja situação era igual a "pendente") no histórico.

O requisito 7 foi considerado essencial ao domínio de Gestão de Recursos de Negócios. Nesse requisito, relacionado ao sistema de reparo de buracos proposto por Pressman (2005), o recurso é o buraco, que é mantido pelo departamento de reparos da prefeitura, mas o dano em si é apenas um registro que não é considerado pelos tipos de gerenciamento tratados pela linguagem de padrões GRN. Esse requisito foi adicionado ao Histórico de Requisitos para que pudesse ser implementado em momento oportuno.

O requisito 8, de controle de segurança, também foi analisado, embora já se soubesse desde a implementação do GREN que seria um requisito desejável para o domínio, mas não implementado. Esse requisito também foi adicionado ao Histórico de Requisitos para que pudesse ser implementado em momento oportuno. Esse é um requisito que deve estar presente em praticamente todas as aplicações do domínio.

A mesma análise ocorreu com o requisito 9, relacionado ao desempenho. Sua ausência foi notada durante a reengenharia do sistema legado de vendas (Silva, 2004a), realizado por um aluno de iniciação científica, com o apoio do PARFAIT. O baixo desempenho do novo sistema foi observado quando a base de dados do sistema legado foi migrada para a do novo sistema e os registros, principalmente os relacionados aos produtos cadastrados (em torno de quarenta mil), tiveram que ser carregados na interface gráfica. Com a análise realizada, concluiu-se que corrigindo esse problema no próprio framework, outros sistemas criados a partir de sua instanciação também teriam vantagem com essa evolução e não sofreriam o mesmo problema de desempenho.

Tabela 4.3: Histórico de Requisitos (Cagnin et al., 2004f)

ID	Requisitos	Solução de projeto implementada no sistema	Razão do "porque" o requisito não é essencial	Sistema	Tipo de Manut.	Situação	Versão do fram.
01	Livros podem ter diversos autores.	Implementar os autores de um livro como atributo multivalorado da classe Livro.	Requisito considerado como candidato a ser essencial ao domínio do framework. Como não havia certeza de sua generalidade, optou-se por implementá-lo no sistema.	Biblioteca (reeng)	Perfectiva	pendente	–
02	Livros podem referir-se a mais de um assunto.	Implementar os assuntos de um livro como atributo multivalorado da classe Livro.	idem a solução do requisito 1.	Biblioteca (reeng)	Perfectiva	pendente	–
03	Aluno é matriculado em um curso.	Implementar o curso no qual o aluno está matriculado na classe Aluno, como uma referência à classe Curso, por meio de um tipo enumerado.	idem a solução do requisito 1.	Biblioteca (reeng)	Perfectiva	pendente	–
04	Aluno possui estado civil (solteiro, casado, divorciado, etc), situação (ativo, inativo, desistente).	Implementar o estado civil e a situação do aluno como tipo enumerado com valores pré-definidos.	idem a solução do requisito 1.	Biblioteca (reeng)	Perfectiva	pendente	–
05	A cidade em que um cliente mora está localizada em uma determinada unidade federativa (SP, MG, RJ, ES, etc)	Implementar a unidade federativa da cidade em que um cliente reside como tipo enumerado com valores pré-definidos.	idem a solução do requisito 1.	Oficina Eletrônica (reeng)	Perfectiva	pendente	–
06	Emissão de etiquetas de mala direta.	Implementar as etiquetas de mala direta a partir da criação da classe Etiqueta, contendo dois tamanhos de etiquetas pré-estabelecidos.	idem a solução do requisito 1.	Oficina Eletrônica (reeng)	Perfectiva	pendente	–

Tabela 4.3: Histórico de Requisitos (Cagnin et al., 2004f) (continuação)

ID	Requisitos	Solução de projeto implementada no sistema	Razão do "porque" o requisito não é essencial	Sistema	Tipo de Manut.	Situação	Versão do fram.
07	Um arquivo de danos é criado para conter informação sobre danos relatados por causa de buracos e incluem nome e endereço do cidadão.	Criar uma classe danos a partir da aplicação sintática do padrão 6 da GRN - *Comercializar o Recurso*, para controlar os danos ocorridos.	Requisito considerado essencial ao domínio do framework, mas optou-se por não implementá-lo devido a decisão do escopo do framework.	Reparos de Buracos (desenv)	Perfectiva	pendente	–
08	Deseja-se autenticar os usuários do sistema, registrar os acessos efetuados e bloquear o acesso a determinadas operações do usuário, conforme o seu papel.	–	Requisito considerado essencial ao domínio do framework desde o desenvolvimento do framework GREN, mas optou-se por não implementá-lo por falta de tempo e devido a decisão do escopo da primeira versão do framework.	–	–	pendente	–
09	Deseja-se resolver o problema de desempenho que ocorre quando grande volume de dados é carregado nas interfaces gráficas de usuário dos sistemas gerados pelo framework.	Perfis foram utilizados para determinar o gargalo do sistema. A carga dos registros na memória é realizada em segundo plano, com baixa prioridade.	Requisito considerado essencial ao domínio do framework mas optou-se por implementá-lo inicialmente no sistema, a fim de propor a melhor solução em um intervalo de tempo menor e com menor custo, já que a hierarquia de classes da aplicação envolve menos classes e menos recursos do que a hierarquia de classes do framework.	Venda (reeng)	Desempenho	pendente	–
...	...	...	...	...	...	...	...

Tabela 4.4: Análise dos requisitos não cobertos pelo framework GREN (Cagnin et al., 2004d)

ID Requisitos	Situações em que ele pode ocorrer novamente
4, 5	Status do pedido em uma fábrica ou loja, status de uma fita em uma locadora, status de um cliente, etc.
3	Especialidade de um médico, convênios atendidos por uma clínica médica, etc.
1, 2	Telefones do cliente, do fornecedor, pessoas de contato de um determinado fornecedor, dependentes do funcionário, doenças associadas a um paciente, etc.
6	Etiquetas de lacre, etiquetas de remetente, etiquetas de patrimônio, etc.
7	A maioria dos sistemas do domínio de Gestão de Recursos de Negócios necessita registrar informação sobre os eventos que ocorrem com um determinado recurso.
8	A maioria dos sistemas do domínio de Gestão de Recursos de Negócios necessita de controle de acesso.
9	A maioria dos sistemas do domínio de Gestão de Recursos de Negócios necessita de processamento rápido.

A partir da análise realizada, observou-se que os requisitos armazenados no Histórico podem ocorrer mais do que duas vezes no desenvolvimento e na reengenharia de sistemas no domínio de Gestão de Recursos de Negócios, portanto são funcionalidades inerentes a esse domínio e sua ausência no framework pode limitar o seu uso. Assim, são candidatos a serem implementados no framework. O requisito 7 não foi incorporado ao framework devido a decisão de escopo. Os requisitos 1 a 6 e os requisitos 8 e 9 foram incorporados ao framework GREN por meio de algumas evoluções. No entanto, somente as evoluções do framework GREN que utilizaram o processo PREF (Cagnin et al., 2004d,f; Silva, 2004b) serão comentadas nesta seção (ou seja, evoluções referentes aos requisitos 1 a 6 e ao requisito 8).

Atividade 4 - Projetar, modificar e testar o framework para satisfazer o requisito
Atividade 3 - Atualizar o histórico de requisitos
Atividade 7 - Validar o framework

As soluções de projeto e as implementações que foram feitas diretamente nos sistemas criados a partir da instanciação do framework GREN para atender requisitos não cobertos por ele (como é o caso dos requisitos agrupados nas seis primeiras linhas do Histórico de Requisitos, Tabela 4.3) foram utilizadas como base para apoiar a evolução do GREN. As soluções de projeto (descritas no Histórico de Requisitos) foram abstraídas e generalizadas e apoiaram no projeto e na implementação de cada evolução do framework. A generalização foi feita a partir do conhecimento do engenheiro do domínio na hierarquia de classes, na arquitetura e no domínio do framework GREN.

Antes de evoluir o framework, tomou-se o cuidado de tratar do Gerenciamento de Controle de Configuração (atividade 8), embora feito de forma manual.

A versão original do framework GREN (versão 1.0) (Braga, 2003) foi submetida a duas evoluções em paralelo. Ambas evoluções resultaram respectivamente nas versões 1.1 e 1.2.

A primeira evolução (Cagnin et al., 2004d,f) foi do tipo perfectiva e, inicialmente, três requisitos não atendidos pelo framework GREN (representados pelos requisitos 1 a 5 do Histórico de Requisitos) foram satisfeitos. Tais requisitos são relacionados a tipos especiais de dados, ou seja, tipo enumerado com valores pré-definidos e com valores obtidos de outra classe, e tipo multivalorado com valores obtidos de outra classe. O Histórico de Requisitos foi atualizado, alterando-se a `situação` dos requisitos 1 a 5 para "sendo atendido" (atividade "Atualizar o histórico de requisitos").

A implementação da primeira evolução do GREN foi relativamente simples de ser feita e exigiu mudanças tanto na estrutura fixa quanto na estrutura variável do framework, pois foram adicionadas novas classes concretas e abstratas para facilitar a inclusão de atributos com tipos especiais nas classes dos sistemas gerados a partir do framework. Notações especiais foram criadas e sugeridas para serem incorporadas à linguagem de padrões GRN, a fim de representar os novos tipos de dados. A documentação de análise e de projeto do framework e o seu *cookbook* foram atualizados (atividade "Atualizar a documentação do framework").

Outro requisito não coberto pelo framework GREN foi também atendido pela primeira evolução. Esse requisito é o de número 6 do Histórico de Requisitos e inclui um novo tipo de saída de relatório, ou seja, etiquetas. O Histórico de Requisitos foi atualizado, alterando-se a `situação` do requisito 6 para "sendo atendido" (atividade "Atualizar o histórico de requisitos"). A implementação desse requisito no framework GREN implicou em mudanças nas estruturas fixa e variável desse framework, pois uma classe concreta foi criada na parte fixa e tornaram-se disponíveis dois tipos de etiquetas na camada variável. A opção desse novo tipo de relatório foi sugerida para ser incluída na linguagem de padrões GRN. A documentação de análise e de projeto do framework e o *cookbook* também foram atualizados (atividade "Atualizar a documentação do framework").

A segunda evolução do GREN (Silva, 2004b) também foi do tipo perfectiva e o requisito relacionado ao controle de acesso (representado pelo requisito 8 do Histórico de Requisitos) foi incorporado ao framework GREN. O Histórico de Requisitos foi atualizado, alterando-se a `situação` do requisito 8 para "sendo atendido" (atividade "Atualizar o histórico de requisitos"). A funcionalidade desse requisito (autenticação, registro de acesso e de bloqueio) foi implementada com o apoio da abordagem orientada a aspectos (Kiczales et al., 1997). Dessa forma, uma série de classes de *AspectS* [4] (Hirschfeld, 2003) foi incorporada à hierarquia do framework GREN. Além disso, foram necessárias grandes modificações na estrutura fixa do framework, uma vez que dezessete classes foram criadas e algumas modificações simples foram feitas na estrutura variável. Em seguida, a documentação de análise e de projeto

[4] AspectS é uma extensão da linguagem Smalltalk, que tem por objetivo dar suporte à Programação Orientada a Aspectos.

do framework e o *cookbook* foram atualizados (atividade "Atualizar a documentação do framework").

Após cada modificação realizada no framework GREN testes de unidade foram realizados. No entanto, nenhum critério de teste específico foi utilizado. Concomitantemente aos testes de unidade, testes de regressão foram feitos na atividade "Validar o framework" por meio de instanciações do framework. Nessas instanciações foram criados sistemas que possuíam os requisitos incorporados ao GREN, para verificar se nenhuma parte do framework havia sido danificada e se os requisitos tinham sido implementados corretamente. Os sistemas tomados como base para as instanciações foram aqueles registrados no Histórico de Requisitos e que possuíam os requisitos incorporados nas duas evoluções do framework.

A instanciação do framework foi feita de duas maneiras: caixa-branca e caixa-preta. A primeira permitiu, além de validar o framework, validar a atualização do *cookbook* feita na atividade "Atualizar a documentação do framework". O segundo tipo de instanciação permitiu validar o framework e também a ferramenta de instanciação GREN-Wizard, atualizada na atividade "Atualizar ferramentas associadas ao framework".

O Histórico de Requisitos foi atualizado, atribuindo "atendido" ao atributo `situação` de cada requisito que foi incorporado ao framework (atividade "Atualizar o histórico de requisitos").

Atividade 5) Atualizar a documentação do framework

Quanto às evoluções no framework GREN, as manutenções perfectivas resultaram em mudanças na documentação do framework (diagrama de classes, documentação da hierarquia de classes, etc) e no processo de instanciação descrito no *cookbook*. Foram descritos os procedimentos necessários para adicionar os novos atributos dos tipos enumerado e multivalorado às classes dos sistemas criados pelo framework. A possibilidade de escolher mais um tipo especial de relatório e a de decidir se uma instanciação adota ou não os requisitos de segurança foram documentadas também no *cookbook*. Esse último caso funciona como se um novo padrão, cuja aplicação é opcional, tivesse sido adicionado à linguagem de padrões.

Atividade 6) Atualizar ferramentas associadas ao framework

A ferramenta de instanciação GREN-Wizard foi evoluída após cada evolução do framework GREN e sua evolução foi baseada no *cookbook* atualizado, o qual indicou indiretamente quais partes do código fonte da ferramenta deveriam ser modificadas. O manual de instanciação da ferramenta GREN-Wizard com as novas funções apresentadas na sua interface gráfica com o usuário (GUI), bem como o diagrama de classes de projeto da ferramenta GREN-Wizard foram atualizados.

A preocupação de tratar do Gerenciamento de Controle de Configuração (atividade 8) também foi considerada antes da evolução da ferramenta de instanciação GREN-Wizard. Igualmente ao do framework GREN, tal gerenciamento também foi feito de forma manual.

Atividade 8) Tratar do Gerenciamento de Controle de Configuração

Como comentado anteriormente, o Gerenciamento de Controle de Configuração do framework GREN e da ferramenta GREN-Wizard foi feito de forma manual, sem apoio de ferramenta computacional específica. Como as evoluções foram realizadas concomitantemente, existem diversas versões em paralelo, conforme apresentado na Figura 4.2. As versões da ferramenta GREN-Wizard são compatíveis com as suas versões correspondentes do framework GREN. O Histórico de Requisitos também foi atualizado nesta atividade, atribuindo o número da versão do framework, em que cada requisito foi incorporado, no atributo `versão do framework`. As linhas do Histórico de Requisitos correspondentes aos requisitos 1 a 6 foram atualizadas, atribuindo-se 1.1 ao atributo `versão do framework`, enquanto que a linha correspondente ao requisito 8 foi atualizada atribuindo-se a versão 1.2.

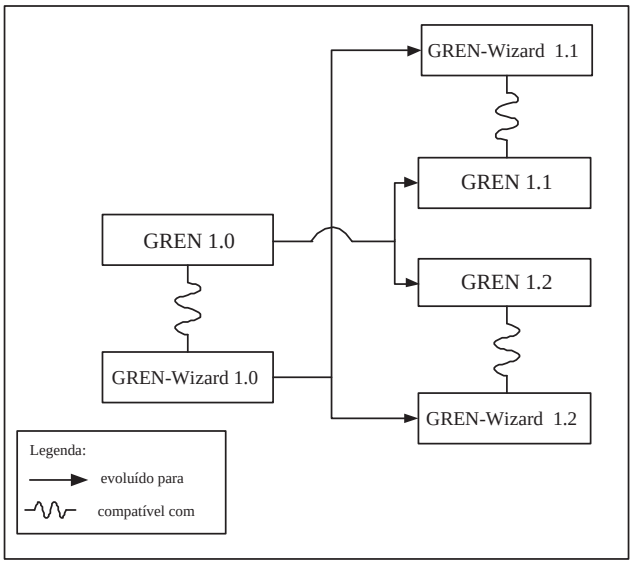


Figura 4.2: Grafo do controle de versões do GREN e da ferramenta GREN-Wizard

Embora exista a necessidade de manter um registro dos requisitos de cada versão do framework com os respectivos sistemas que foram gerados em cada versão, conforme ilustrado na Figura 4.3, isso ainda não foi feito. No entanto, é uma preocupação que deve ser considerada, pois caso um sistema seja importado em uma hierarquia de classes do framework cuja versão é diferente daquela na qual ele foi gerado, pode haver conflito entre o código fonte do framework e o do sistema e, conseqüentemente, o sistema pode não fornecer o comportamento desejado.

Como o GREN é um framework acadêmico, diversos pesquisadores estão utilizando-o e, portanto, pode ser alterado a qualquer momento, por qualquer pesquisador, gerando

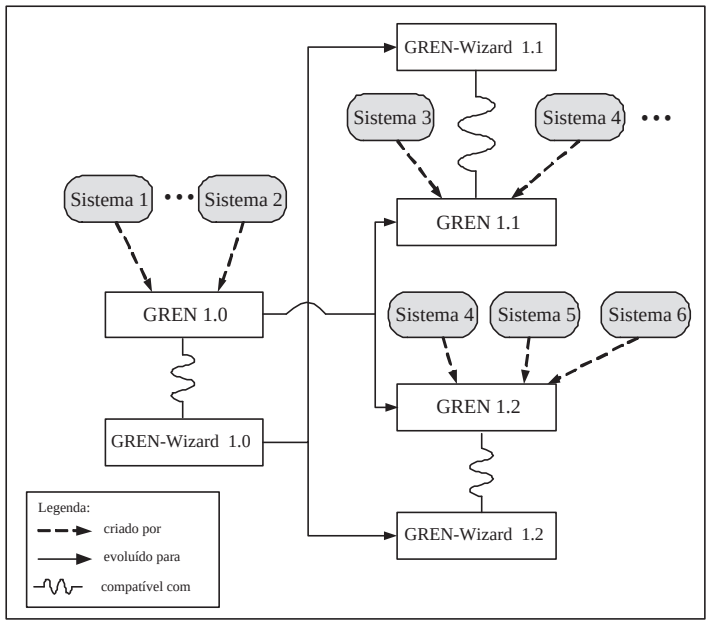


Figura 4.3: Grafo do controle de versões do GREN e dos sistemas gerados

diversas versões em paralelo e dificultando ainda mais o controle de suas versões. Dessa forma, o ideal é criar um repositório em um ferramenta de controle de versões, por exemplo, CVS (Concurrent Versions System, 2004), para facilitar a integração do código fonte do framework, que deve ser feita constantemente. O pesquisador, antes de submeter a nova versão do framework à ferramenta, deve integrá-la à última versão do framework que está disponível no repositório e garantir que ela esteja funcionando adequadamente para que outros possam utilizá-la.

4.4 Considerações Finais

Neste capítulo apresentou-se um processo de evolução de frameworks de aplicação (PREF), que é um dos recursos criados e associados ao arcabouço de reengenharia definido nesta tese. Apesar desse processo ter alguns passos que são similares a qualquer outro processo de manutenção de software convencional, fornece contribuição relacionada ao controle de variabilidade de framework. Tal controle apóia o engenheiro de domínio decidir se um requisito não coberto pelo framework (dito variabilidade requerida por Deelstra et al. (2004)) é essencial ou não ao seu domínio, ou seja, deve ser incorporado ao framework. Para isso, utiliza um Histórico de Requisitos que:

- apóia a decisão se o requisito é essencial ao domínio do framework;
- documenta os requisitos não cobertos pelo framework e, como mantém a documentação daqueles incorporados ao framework, é possível consultar quais os *hot spots* do framework foram incorporados e/ou alterados em uma determinada versão;

- pode apoiar a implementação dos requisitos no framework quando esses foram implementados anteriormente em sistemas gerados a partir da instanciação do framework, pois descreve a solução de projeto adotada.

O principal benefício do controle de variabilidade do PREF é a identificação de novos *hot spots* e/ou a identificação de *hot spots* existentes que devem ser alterados para permitir que o framework seja instanciado para um número maior de sistemas pertencentes ao seu domínio.

Um dos cinco objetivos pré-definidos para utilizar o método de avaliação de variabilidade de software COSVAM (Deelstra et al., 2004) é similar ao objetivo do controle de variabilidade do PREF, ou seja, "durante a derivação do produto, determinar se as características não fornecidas devem ser implementadas no produto ou ser integradas à família de produtos". No entanto, além de fornecer atividades para apoiar a decisão de evolução do framework, o processo proposto também fornece atividades específicas para apoiar tanto a evolução do framework quanto a dos sistemas gerados a partir dele (se for o caso). Tanto o processo proposto como a técnica COSVAM consideram *hot spots* como elementos centrais para o gerenciamento de variabilidade. No COSVAM o gerenciamento é feito observando o desacordo entre a variabilidade fornecida e a variabilidade requerida e determinando quais mudanças são requeridas para resolvê-lo e quais os efeitos dessas mudanças na família de produtos. No PREF o gerenciamento é feito principalmente verificando, por meio de algumas diretrizes, se a variabilidade requerida pertence ao domínio do framework e se sua ausência pode limitar a instanciação do framework para um número significativo de sistemas, ou se a variabilidade requerida não pertence ao domínio do framework. Quando a variabilidade requerida pertence ao domínio do framework, é necessário analisar viabilidades técnica, financeira, de escopo, entre outras, para apoiar a decisão de incorporá-la ou não a ele.

Embora o PREF tenha sido utilizado apenas para apoiar manutenções perfectivas de um determinado framework, acredita-se que atividades de manutenção dos tipos corretiva, adaptativa, preventiva, redutiva e de desempenho (Chapin et al., 2001) também possam ser conduzidas aplicando-se as atividades 3 a 8.

O processo de evolução proposto, apesar de ter sido efetivo nos contextos em que foi aplicado, ainda não pode ser considerado genérico, pois a experiência com evoluções em um framework pertencente ao domínio de Gestão de Recursos de Negócios, apresentada na Seção 4.3, não é suficiente para validar e garantir a generalidade de tal processo. É necessário aplicá-lo nas evoluções de outros frameworks em outros domínios e, principalmente, em frameworks comerciais para que possa ser validado e aprimorado.

Outra limitação é o Gerenciamento de Controle de Configuração mais sistemático das versões do framework e das ferramentas de apoio (caso existam) com as respectivas versões dos sistemas gerados.

No próximo capítulo apresenta-se a abordagem ARTe, que associa recursos de teste funcional a linguagem de padrões de análise, mais especificamente a GRN, objetivando proporcionar reúso de recursos de teste tanto no desenvolvimento quanto na reengenharia quando se usa esse tipo de linguagem de padrões. Apresenta-se também a aplicabilidade dessa abordagem na linguagem de padrões GRN. Essa abordagem também está disponível no arcabouço ARA definido no capítulo anterior.

Capítulo

5

ARTe: Uma Abordagem de Reúso de Teste baseada na Linguagem de Padrões de Análise GRN

5.1 Considerações Iniciais

Neste capítulo apresenta-se uma Abordagem de Reúso de Teste (ARTe), proposta nesta tese, e foi associada ao arcabouço de reengenharia ágil ARA para permitir reúso na reengenharia no nível de teste. Essa abordagem captura aspectos de validação das soluções embutidas nos padrões de linguagens de padrões de análise que possuem as seguintes características: modelam sistemas de informação que possuem uma arquitetura composta de três camadas, sendo elas, 1) interface com o usuário (entrada e saída de dados), 2) regras de negócio e 3) armazenamento de dados em banco de dados relacional, como comentado no Capítulo 1. A proposição dessa abordagem foi motivada pelos resultados de um estudo de caso de reengenharia utilizando o PARFAIT (Capítulo 3), em que se observou que grande parte do tempo e do esforço da reengenharia estava sendo despendido em atividades de VV&T, como relatado na Seção 3.5 do Capítulo 3. Isso também ocorre no desenvolvimento de software e é ressaltado por diversos autores (Harrold, 2000; Pressman, 2005; Rocha et al., 2001; Sommerville, 2000). A abordagem proposta permite a agregação e o reúso de recursos de teste funcional (classes de equivalência, requisitos e casos de teste) às linguagens de padrões de análise, mais especificamente, à linguagem de padrões GRN (Braga et al., 1999).

A elaboração dos recursos de teste funcional na abordagem proposta é baseada nos critérios de teste Particionamento de Equivalência e Análise do Valor Limite (Myers, 2004a), que são utilizados por serem os mais conhecidos e difundidos na literatura. As classes de equivalência, os requisitos e os casos de teste criados ficam disponíveis para serem reutilizados tanto durante a reengenharia quanto durante o desenvolvimento de software quando se usam linguagens de padrões de análise. Para isso, diretrizes específicas são fornecidas pela abordagem.

As diretrizes de reúso de teste foram utilizadas em um estudo de caso de reengenharia apresentado neste capítulo,utilizando o arcabouço de reengenharia proposto. Apesar de terem sido reutilizados somente parte dos recursos de teste disponíveis, houve uma redução de tempo em torno de 57% nas atividades de teste. Os dados obtidos no estudo de caso não podem ser considerados conclusivos, sendo necessário conduzir experimentos controlados para analisar estatisticamente os resultados obtidos com a aplicação das diretrizes de reúso da abordagem proposta e comprovar a redução de tempo obtida.

Este capítulo está organizado da seguinte forma: na Seção 5.2 é apresentada a abordagem ARTe. Na Seção 5.3 apresenta-se um exemplo de uso da abordagem ARTe em que recursos de teste funcional são criados e agregados a alguns padrões da linguagem de padrões GRN. Nessa seção apresenta-se também um estudo de caso de reengenharia que reutiliza, por meio das diretrizes de reúso da abordagem proposta, alguns recursos de teste disponíveis com o apoio do arcabouço definido. Esse estudo de caso foi conduzido para analisar a redução do tempo da reengenharia com reúso de requisitos de teste em relação a um outro estudo de caso sem reúso, discutido na Seção 3.5 do Capítulo 3. Na Seção 5.4 são apresentadas as considerações finais referentes a este capítulo.

5.2 A Abordagem de Reúso de Teste ARTe

A abordagem ARTe (Cagnin et al., 2005a, 2004c) é composta por duas etapas, como apresentado na Figura 5.1. A Etapa 1 consiste de uma estratégia que apóia a definição de requisitos e casos de teste funcional para cada padrão da linguagem de padrões de análise (LPA) e os disponibiliza em seções específicas da documentação dessa linguagem. Esses requisitos são criados com o apoio dos critérios de teste Particionamento de Equivalência e Análise do Valor Limite, e os casos de teste são derivados a partir dos requisitos definidos. A Etapa 2 consiste de diretrizes que apóiam o reúso das classes de equivalência, dos requisitos e dos casos de teste criados pela estratégia. Dessa forma, é possível minimizar os esforços e custos despendidos nas atividades de VV&T, tanto no desenvolvimento quanto na reengenharia de software, quando se usa uma LPA. Isso porque o engenheiro de software reutiliza não somente as soluções de análise dos padrões, mas também as informações de teste disponíveis.

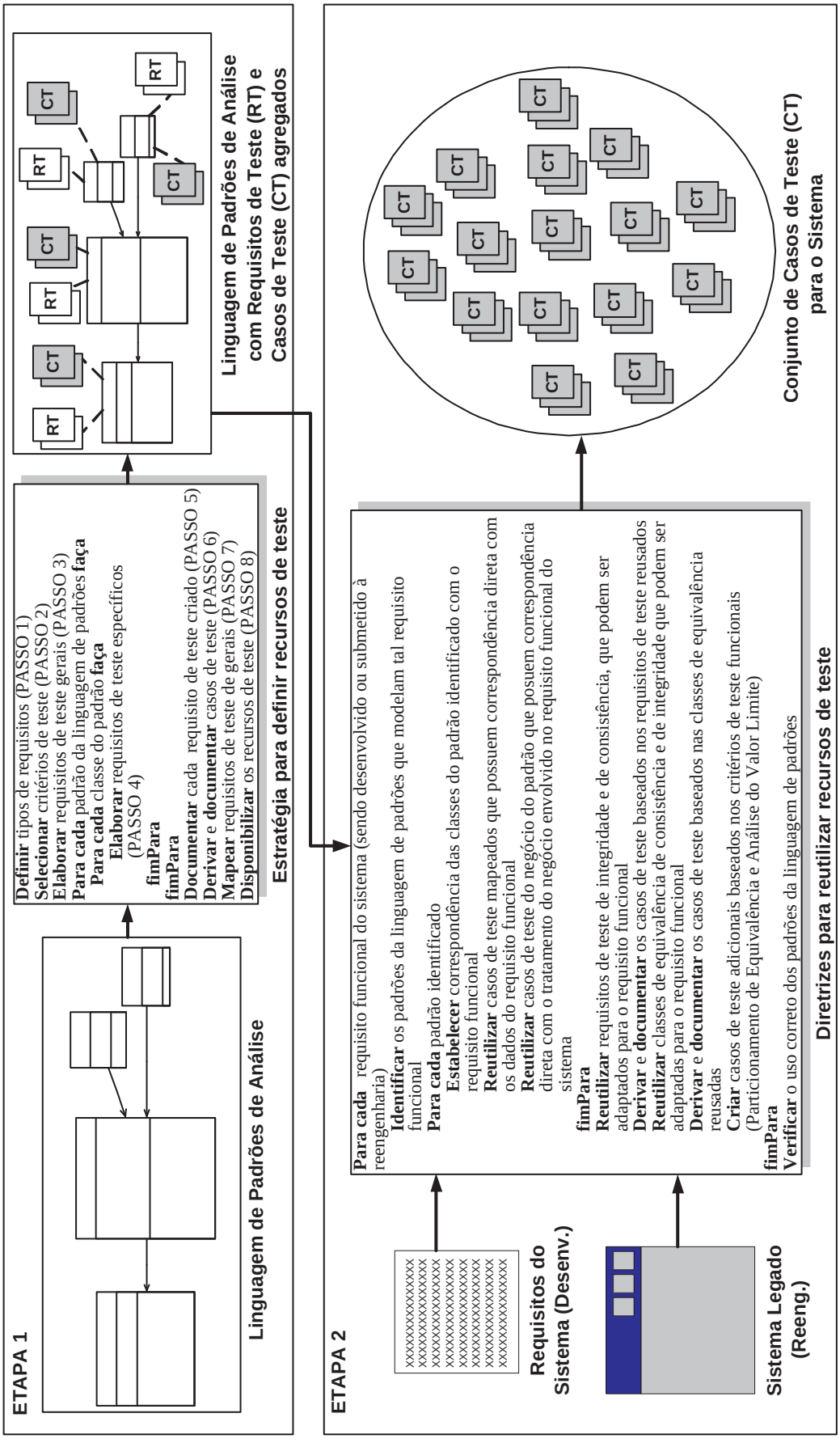


Figura 5.1: Visão geral da abordagem ARTe (adaptada de Cagnin et al. (2004c))

Resumidamente, na Etapa 1, dada uma LPA, aplicam-se os passos da estratégia da abordagem, resultando em uma LPA com recursos de teste associados aos padrões da linguagem. Essa estratégia pode ser aplicada durante a criação da linguagem, por exemplo, retomando informações de VV&T consideradas pelos sistemas que motivaram as soluções representadas nos padrões. Na Etapa 2, dado um projeto de desenvolvimento ou de reengenharia com o apoio de uma LPA com recursos de teste associados, aplicam-se as diretrizes de reúso da abordagem, resultando em um conjunto de casos de teste para o sistema.

5.2.1 Estratégia para Definir Recursos de Teste

Na Tabela 5.1 apresenta-se uma visão geral da estratégia da abordagem ARTe, no contexto de linguagens de padrões em diversos níveis de abstração. Na Tabela 5.2 apresenta-se uma instanciação da estratégia apresentada na Tabela 5.1, mais especificamente no contexto de LPAs que modelam sistemas de informação com arquitetura em três camadas, de acordo com o que foi especificado na Seção 5.1 deste capítulo, como é o caso da linguagem de padrões GRN. Os passos destacados na Tabela 5.2 referem-se a passos específicos para o contexto de LPAs. Uma descrição de cada passo da estratégia, bem como dos passos específicos para o contexto de LPAs, é apresentada a seguir.

Tabela 5.1: Estratégia para associar recursos de teste a linguagens de padrões

- **Definir** tipos de requisitos (PASSO 1)
- **Selecionar** critérios de teste existentes (PASSO 2)
- **Elaborar** requisitos de teste comuns (PASSO 3)
- **Para** cada padrão da linguagem de padrões
  - **Para** cada classe de cada padrão da linguagem de padrões
    - **Elaborar** requisitos de teste específicos (PASSO 4)
  - **FimPara**
- **FimPara**
- **Documentar** cada requisito de teste criado (PASSO 5)
- **Derivar e documentar** casos de teste (PASSO 6)
- **Mapear** casos de teste comuns (PASSO 7)
- **Disponibilizar** os recursos de teste (PASSO 8)

Inicialmente, durante a criação da estratégia observou-se a necessidade de identificar os tipos de requisitos (**PASSO 1**), de acordo com os aspectos do software que devem ser considerados pelos testes, por serem importantes na verificação e validação do sistema. Isso depende do domínio e do interesse em que o padrão está inserido. Nesse primeiro passo

Tabela 5.2: Estratégia para associar recursos de teste a LPAs

- **Definir** tipos de requisitos (PASSO 1)
- **Selecionar** critérios de teste existentes (PASSO 2)
- **Elaborar** requisitos de teste de consistência (PASSO 3.1)
- **Elaborar** requisitos de teste de integridade (PASSO 3.2)
- **Para** cada padrão da LPA
  - **Para** cada classe de cada padrão da LPA
    - **Elaborar** requisitos de teste do negócio (PASSO 4)
  - **FimPara**
- **FimPara**
- **Documentar** cada requisito de teste criado (PASSO 5)
- **Derivar e documentar** casos de teste (PASSO 6)
- **Mapear** casos de teste comuns (PASSO 7)
- **Disponibilizar** os recursos de teste (PASSO 8)

foram definidos três tipos de requisitos de teste, necessários para testar as características de interesse da arquitetura do sistema baseado em padrões de análise: de consistência, de integridade e do negócio, visando criar um conjunto de teste funcional mais completo possível no domínio de Sistemas de Informação.

A criação do **requisito de teste de consistência** foi motivada pela importância de validar os dados do sistema antes de serem armazenados. O requisito de teste de consistência preocupa-se com a consistência dos dados do sistema, como: tamanho, valor (mínimo e máximo), tipo (`integer`, `float`, `string`, `date`, entre outros), obrigatório, entre outros.

A definição do **requisito de teste de integridade** foi motivada pela importância do armazenamento físico dos dados processados pelos sistemas de informação em bancos de dados relacionais, e, principalmente, pela necessidade de garantir a integridade no armazenamento dos dados para que, posteriormente, possam ser recuperados corretamente. Esse requisito de teste é limitado às características chave primária e chave estrangeira de bancos de dados relacionais, portanto não considera outras características como *triggers* e *stored procedures*. O requisito de teste de integridade preocupa-se principalmente em verificar a existência de valor único de chave primária e a existência da chave primária, estabelecida como valor de chave estrangeira em uma ou mais tabelas associadas.

Ressalta-se que existem outras formas de armazenamento físico de dados (por exemplo, arquivos texto, bancos de dados orientados a objetos, etc), mas não são tratadas pela abordagem proposta.

O **requisito de teste do negócio** tem como objetivo garantir o correto tratamento das regras do negócio e do funcionamento do sistema. Esse requisito é baseado nas

particularidades do funcionamento do negócio e preocupa-se com as funções do negócio embutidas em cada padrão da linguagem de padrões.

Em seguida, para apoiar a criação de cada tipo de requisito de teste, é necessário decidir quais critérios de teste existentes serão utilizados (**PASSO 2**). Inicialmente, o tipo de critério que será utilizado deve ser estudado. Critérios de teste funcional devem ser selecionados quando a especificação do produto é considerada para derivar os requisitos de teste, por exemplo, quando o padrão é de análise. Por outro lado, quando a implementação do produto é considerada, critérios de teste estrutural devem ser selecionados, por exemplo, quando o padrão é de projeto ou de implementação. Nesse caso, o oráculo pode ser o cliente ou o próprio comportamento do sistema legado. Depois de selecionar o tipo de critério, é necessário observar os objetivos, custo, eficácia e *strength* [1] dos critérios do tipo selecionado para que possam ser corretamente escolhidos.

No contexto de LPAs com características de interesse desta tese, foram selecionados os critérios de teste funcional Particionamento de Equivalência e Análise do Valor Limite (Myers, 2004a), e utilizou-se a idéia geral da técnica "Teste de Validação de Entrada" (Hayes e Offutt, 1999), ou seja, identificar os dados de teste que tentam mostrar a presença ou a ausência de falhas específicas, pertinentes à tolerância a entradas (ou seja, verifica a habilidade do sistema processar adequadamente valores de entrada esperados e inesperados).

Após a criação dos passos de definição dos tipos de requisitos e da seleção dos critérios de teste, foram definidos os outros passos da estratégia, apresentados na Tabela 5.1, que apóiam a criação de cada tipo definido de requisito e a derivação dos respectivos casos de teste. Os recursos de teste criados com a aplicação da estratégia são refinados tanto devido à evolução da própria LPA quanto devido ao seu uso durante o desenvolvimento e a reengenharia de software.

No **PASSO 3** (Elaborar requisitos de teste comuns) devem-se criar requisitos de teste, baseados em cada tipo de requisito definido no **PASSO 1**, e que podem ser considerados para testar todos os padrões da linguagem de padrões. A criação desses requisitos deve seguir as diretrizes dos critérios de teste selecionados no **PASSO 2**.

No caso da instanciação da estratégia para LPAs, os requisitos de teste de consistência, de integridade e do negócio são criados com base nas diretrizes dos critérios de teste funcional Particionamento de Equivalência e Análise do Valor Limite.

No **PASSO 3.1** (Criar requisitos de teste de consistência) (Tabela 5.2) deve-se definir classes de equivalência, relacionadas à consistência, que são comuns à maioria dos padrões de análise. Diretrizes são fornecidas para guiar a criação das classes de equivalência de consistência, sendo descritas a seguir.

Para criar as classes de equivalência de consistência, considerar as classes válidas e inválidas para cada tipo de dado básico (por exemplo, `integer`, `float`, `string` e `date`)

[1] *Strength* significa dificuldade de satisfação, que se refere à dificuldade de satisfazer um critério já tendo satisfeito um outro (Maldonado e Fabbri, 2001a).

e também a partir de outros tipos de dados (por exemplo, vetor, enumerado, multivalorado, entre outros), que são considerados nos padrões da LPA, levando em conta as operações de manipulação de dados, ou seja, inserção, alteração, remoção e consulta. Por exemplo, nas operações de inserção e alteração é necessário verificar a consistência de todos os atributos da classe, para que nenhuma informação incorreta seja armazenada no banco de dados.

Para atributos que são do tipo `integer` ou `float`, criam-se classes de equivalência válidas e inválidas que consideram valor máximo e mínimo que o atributo pode ter, valor do atributo dentro do intervalo válido estabelecido, valor nulo (ou zero) para o atributo, tamanho máximo e mínimo do atributo e tamanho do atributo dentro do limite determinado. Para atributos do tipo `string` criam-se classes de equivalência válidas e inválidas que consideram tamanho mínimo e máximo do atributo, tamanho do atributo dentro do limite estabelecido e valor do atributo nulo (vazio). Por outro lado, para atributos do tipo `date`, criam-se também classes de equivalência válidas e inválidas, mas que consideram, por exemplo, dia com valor maior que o número de dias permitido para o mês, mês maior que 12, ano igual a zero, ano com a formatação estabelecida (2 dígitos ou 4 dígitos).

Para atributos que tenham tipo diferente dos tipos básicos mencionados anteriormente, é necessário verificar as possíveis classes de equivalência válidas e inválidas que podem ser abstraídas a partir das sugestões estabelecidas para os tipos `integer`, `float`, `string` e `date` e criar outras classes que considerem características específicas do tipo do atributo.

Na Tabela 5.3 apresenta-se o gabarito da documentação das classes de equivalência, adaptada da sugerida por Myers (2004a). Os números das classes de equivalência são apresentados entre parênteses.

Tabela 5.3: Gabarito da documentação das classes de equivalência de consistência e de integridade

Operação	**Tipo do Requisito**	**Classes Válidas**	**Classes Inválidas**
Operação de manipulação de dados considerada pela classe de equivalência	Tipo do requisito (de consistência ou de integridade) que a classe de equivalência está relacionada	Especificação das classes de equivalência válidas **(1, 2)**, ...	Especificação das classes de equivalência inválidas **(3, 4)**, ...
...	...	...	...

No **PASSO 3.2** (Criar requisitos de teste de integridade) (Tabela 5.2) deve-se definir classes de equivalência, relacionadas à integridade de dados, que são comuns à maioria dos padrões de análise. Para criar essas classes de equivalência deve-se considerar as classes válidas e inválidas para cada regra de integridade de bancos de dados relacionais. Por exemplo, na operação de remoção é necessário que a informação exista para que possa ser removida, então a classe válida é "valor cadastrado" e a classe inválida é "valor não cadastrado". Além disso, a informação não pode ter relacionamentos pendentes, ou seja, não pode estar cadastrada como chave estrangeira em tabelas relacionadas. Em operações de inserção, é necessário que a informação ainda não exista para que possa ser inserida, então a classe válida é "valor não cadastrado" e a classe inválida é "valor cadastrado". Por outro

lado, em operações de alteração e consulta, é necessário que a informação exista para que possa ser alterada ou consultada, então a classe válida é "valor cadastrado" e a classe inválida é "valor não cadastrado".

A criação dos requisitos de teste de consistência e de integridade é baseada nas classes de equivalência definidas (válidas e inválidas), de consistência e de integridade, respectivamente, e no critério Análise do Valor Limite, observando os limites logo acima e logo abaixo de cada classe de equivalência. Detalhes de como documentar os requisitos de teste são comentados no **PASSO 5**.

No **PASSO 4** (Criar requisitos de teste do negócio) devem-se criar classes de equivalência válidas e inválidas a partir de funções específicas do negócio, incorporadas em cada padrão da LPA (regras do negócio do domínio ao qual a LPA pertence), considerando suas condições. Por exemplo, em uma locação, o recurso só pode ser locado se estiver disponível no momento. Posteriormente, criam-se requisitos de teste baseados nas classes de equivalência e no critério Análise do Valor Limite, considerando as classes de equivalência definidas.

A documentação das classes de equivalência do requisito de teste do negócio foi também adaptada da proposta por Myers (2004a) e contém informações adicionais àquela apresentada na Tabela 5.3, referentes à classe do padrão que o requisito pertence, à tabela correspondente no SGBD, ao atributo da classe do padrão envolvido na regra do negócio e uma observação, que indica quando o requisito de teste deve ser considerado, caso necessário, como ilustrado na Tabela 5.4. Ressalta-se que os números das classes de equivalência de consistência, de integridade e do negócio devem ser distintos.

Tabela 5.4: Gabarito da documentação das classes de equivalência do negócio

Operação	**Classe do Padrão**	**Tabela Correspondente**	**Atributo**	**Classes Válidas**	**Classes Inválidas**	**Observação**
Operação de manipulação de dados considerada pela classe de equivalência	Classe do padrão considerada pela classe de equivalência	Tabela correspondente à classe do padrão	Atributo da classe do padrão considerado pela classe de equivalência	Especificação das classes de equivalência válidas (**5, 6, ...**)	Especificação das classes de equivalência inválidas (**7, ...**)	Observação que indica em quais situações (i.e. variantes do padrão) a classe de equivalência válida ou inválida deve ser considerada
...	...	...	...	...	...	...

No **PASSO 5** (Documentar cada requisito de teste criado), documentam-se todos os tipos de requisitos criados. Para isso, sugere-se que a documentação dos requisitos de teste comuns à maioria dos padrões (por exemplo, requisitos de teste de consistência e de integridade) contenha as seguintes informações e que sejam apresentadas em formato tabular:

- **número** do requisito de teste: número que identifica unicamente o requisito de teste;
- **tipo** do requisito de teste: classifica o tipo do requisito de teste, de acordo com os tipos definidos no PASSO 1;

- número das **classes de equivalência**: número das classes de equivalência utilizadas para criar o requisito de teste;
- tipo de **operação** de manipulação de dados tratado pelo requisito de teste (ou seja, inserção, alteração, remoção ou consulta): obtida da documentação das classes de equivalência utilizadas;
- **condições válidas**: consideradas para analisar o dado de entrada e estabelecer o retorno esperado;
- **especificação** do requisito de teste: descreve o que o dado de entrada do caso de teste a ser derivado no PASSO 6 deve considerar;
- **observação**: fornece alguma informação relevante a respeito do requisito de teste;
- **retorno esperado**: especifica o retorno esperado do requisito de teste.

Com relação à documentação dos requisitos de teste, específicos de cada classe dos padrões da linguagem de padrões (como é o caso dos requisitos do negócio), adicionam-se as seguintes informações em relação à documentação dos requisitos de teste comuns à maioria dos padrões: nome do **padrão**; nome da **classe do padrão** e **tabela do SGBD** correspondente à classe que o requisito de teste pertence.

O **PASSO 6** (Derivar e documentar casos de teste) apóia a derivação de casos de teste a partir dos requisitos definidos e apresenta como deve ser feita a documentação de cada caso de teste criado. Neste passo é necessário considerar todos os requisitos de teste criados e derivar um ou mais casos de teste, observando a especificação e as condições válidas do requisito de teste. Sugere-se que a documentação dos casos de teste seja em formato tabular. A documentação de cada caso de teste, derivado a partir dos requisitos de teste comuns à maioria dos padrões, deve conter as informações citadas a seguir, das quais algumas são oriundas da documentação do requisito de teste. Essa redundância visa facilitar a leitura do caso de teste quando for reutilizado.

- **número do caso de teste**: identifica unicamente o caso de teste;
- **número do requisito de teste** (obtido da documentação do requisito de teste);
- tipo de **operação** de manipulação de dados tratado pelo requisito de teste (obtido da documentação do requisito de teste);
- **validações anteriores**: especifica o número dos casos de teste que devem ser considerados como pré-condição;
- **dado de entrada**: especifica o valor que deve ser utilizado como dado de entrada do teste, baseando-se no item "especificação" da documentação do requisito de teste;

- **saída esperada**: especifica o valor esperado como saída do teste em relação ao dado de entrada, baseando-se no item "condições válidas" da documentação do requisito de teste.

Com relação à documentação dos casos de teste derivados a partir dos requisitos de teste específicos, adicionam-se as seguintes informações: nome do **padrão** (obtido da documentação do requisito de teste) e nome da **classe do padrão** (obtido da documentação do requisito de teste).

No **PASSO 7** (Mapear casos de teste comuns) elabora-se um mapeamento dos casos de teste criados, comuns a todos os padrões, para cada atributo de cada padrão da linguagem de padrões para facilitar o reúso. Para fazer esse mapeamento, é necessário saber o tipo e o tamanho de cada atributo de cada classe participante, identificar o atributo de cada classe participante mapeado como chave primária no banco de dados relacional e identificar todos os relacionamentos existentes entre as classes participantes de cada padrão, mapeados como chaves estrangeiras no banco de dados relacional. A documentação do mapeamento deve conter as seguintes informações: nome do **padrão**, nome da **classe do padrão**, **tabela do SGBD** correspondente à classe a que o requisito de teste pertence, nome do **atributo**, **número do(s) caso(s) de teste** e **validações anteriores**.

No **PASSO 8** (Disponibilizar os recursos de teste), a documentação dos recursos de teste específicos é disponibilizada em uma nova seção do padrão ao qual pertence, denominada *Informações VV&T*. Por exemplo, as classes de equivalência específicas do negócio, os requisitos de teste do negócio e os casos de teste derivados a partir desses requisitos são disponibilizados. Nessa seção disponibiliza-se também o mapeamento criado no **PASSO 7**.

A documentação das classes de equivalência, os requisitos e os casos de teste relacionados à consistência e à integridade, por serem aplicáveis a todos os padrões da LPA, são disponibilizados no final da documentação dos padrões da LPA, em uma nova seção criada, também denominada *Informações VV&T*. Em ambas as seções criadas (no padrão e no final da linguagem de padrões), os nomes dos critérios de teste funcional utilizados para a criação dos recursos de teste, por exemplo, Particionamento de Equivalência e Análise do Valor Limite, no caso específico de LPAs, são também divulgados. Essa informação será útil, principalmente, quando houver necessidade de evoluir a linguagem de padrões e, conseqüentemente, as informações de VV&T associadas.

As diretrizes para apoiar o reúso dos recursos de teste associados aos padrões da LPA estão apresentadas na próxima subseção.

5.2.2 Diretrizes para Reutilizar Recursos de Teste

Após a instanciação da estratégia para associar recursos de teste funcional às LPAs, foram especificadas as diretrizes que apóiam o reúso desses recursos de teste. A criação dessas

diretrizes foi apoiada pela experiência na aplicação da linguagem de padrões GRN (Braga et al., 1999) em estudos de caso de desenvolvimento e de reengenharia de software, e foi refinada durante o estudo de caso conduzido em um projeto de iniciação científica e apresentado na Subseção 5.3.2 deste capítulo.

Durante a definição das diretrizes, observou-se que o engenheiro de software poderia usufruir do reúso mesmo quando alguns requisitos funcionais do sistema não fossem correspondentes a algum padrão da LPA. Dessa maneira, é possível reutilizar e adaptar requisitos de teste e classes de equivalência que mais se aproximam do requisito funcional do sistema sendo analisado.

Na Tabela 5.5 são apresentadas as diretrizes que apóiam três níveis de reúso de recursos de teste (classes de equivalência, requisitos e casos de teste) na reengenharia e no desenvolvimento de software, quando se usa LPAs com informações de testes associadas.

Tabela 5.5: Diretrizes para reúso de recursos de teste (adaptadas de Cagnin et al. (2004c))

Para cada requisito funcional do sistema (sendo desenvolvido ou submetido à reengenharia)

- **Identificar** os padrões da LPA que modelam tal requisito funcional
  - **Para cada** padrão identificado
    - **Estabelecer** correspondência das classes do padrão identificado com o requisito funcional
    - **Reutilizar** casos de teste mapeados [a] que possuem correspondência direta [b] com os dados do requisito funcional do sistema
    - **Reutilizar** casos de teste do negócio do padrão que possuem correspondência direta com o tratamento do negócio envolvido no requisito funcional do sistema
  - **FimPara**
- **Reutilizar** requisitos de teste de consistência e de integridade, que podem ser adaptados para o requisito funcional do sistema
- **Derivar** e **documentar** os casos de teste baseados nos requisitos de teste reutilizados
- **Reutilizar** classes de equivalência de consistência e de integridade que podem ser adaptadas para o requisito funcional do sistema
- **Derivar** e **documentar** os casos de teste baseados nas classes de equivalência reutilizadas
- **Criar** casos de teste adicionais baseados nos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite

FimPara
Verificar o uso correto dos padrões da linguagem de padrões

[a] Casos de teste comuns aos padrões da linguagem de padrões, mapeados para os atributos do padrão identificado.
[b] Ou seja, os atributos do padrão são equivalentes aos dados do requisito funcional do sistema.

A documentação recomendada para os casos de teste derivados a partir dos requisitos de teste reutilizados, tanto para requisitos funcionais do sistema com correspondência quanto para requisitos funcionais sem correspondência com os padrões da LPA, é similar àquela apresentada no PASSO 6 da estratégia proposta na Subseção 5.2.1, adicionando-se o nome do requisito funcional do sistema.

Quando o caso de teste, referente ao requisito funcional do sistema sem correspondência com os padrões da LPA, é derivado a partir de classes de equivalência disponíveis, é necessário criar e documentar requisitos a partir dessas classes de equivalência; e, em seguida, derivar e documentar os casos de teste, de acordo com a documentação recomendada no PASSO 5 e no PASSO 6, respectivamente.

Por outro lado, quando o caso de teste é criado diretamente a partir dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite então é necessário: 1) criar e documentar as classes de equivalência necessárias, 2) criar e documentar os requisitos de teste a partir dessas classes de equivalência e 3) derivar e documentar os casos de teste. A documentação dos itens 1, 2 e 3 deve ser feita de acordo com os passos 3.1 e 3.2, 5 e 6, respectivamente.

Finalmente, é necessário verificar se os padrões da LPA foram utilizados corretamente, verificando se a seqüência de aplicação dos padrões foi seguida, observando a estrutura da linguagem de padrões (no caso da linguagem de padrões GRN, a estrutura é representada por um grafo, como ilustrado na Figura 2.2 da página 21) e a seção da documentação *próximos padrões*. Além disso, é necessário verificar se todas as classes participantes e obrigatórias do padrão foram consideradas na modelagem do sistema, observando as seções *estrutura*, *participantes* e *variantes* de cada padrão utilizado.

5.3 Uso da Abordagem ARTe em uma Linguagem de Padrões de Análise

A estratégia e as diretrizes da abordagem ARTe foram aplicadas em alguns padrões da linguagem de padrões GRN e em um estudo de caso de reengenharia, respectivamente. Isso colaborou para o refinamento de ambas.

Na Subseção 5.3.1 apresenta-se um exemplo de uso da estratégia da abordagem ARTe em um padrão da GRN. Na Subseção 5.3.2 relata-se um estudo de caso de reengenharia que foi conduzido utilizando o PARFAIT e, conseqüentemente, o arcabouço ARA. Nesse estudo foram reutilizados classes de equivalência e requisitos de teste funcional, definidos para os padrões da GRN, visando observar a redução do tempo da reengenharia, uma vez que as atividades de VV&T do PARFAIT consumiam grande parte do tempo da reengenharia, como discutido no estudo de caso relatado na Seção 3.5 do Capítulo 3.

5.3.1 Uso da Estratégia de Definição de Recursos de Teste

A estratégia da abordagem ARTe foi aplicada, até o momento da escrita desta tese, nos seguintes padrões da GRN: Padrão 1 – IDENTIFICAR O RECURSO, Padrão 2 – QUANTIFICAR O RECURSO, Padrão 3 – ARMAZENAR O RECURSO, Padrão 4 – LOCAR O RECURSO, Padrão 9 – MANTER O RECURSO e Padrão 13 – IDENTIFICAR O EXECUTOR DA TRANSAÇÃO.

A definição dos requisitos e a derivação dos casos de teste funcional para a classe `Locação de Recurso` do Padrão 4 da GRN, bem como a disponibilização desses recursos de teste, são apresentados nesta subseção. Esse padrão foi escolhido pois é um dos três principais padrões da GRN e trata de uma das principais funcionalidades do domínio de Gestão de Recursos de Negócios. Na Figura 5.2 apresentam-se o diagrama de classes desse padrão e o mapeamento da classe `Locação de Recurso` para o banco de dados relacional MySQL, que é feito automaticamente pelo framework GREN. O *script* da tabela está na língua inglesa, como é originalmente criado pelo framework.

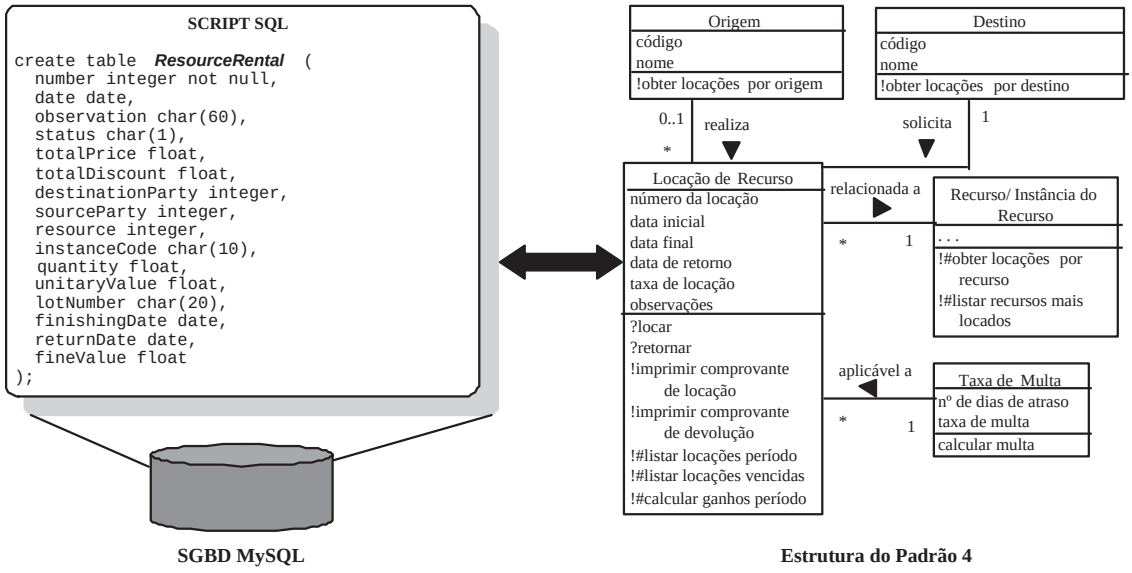


Figura 5.2: Padrão 4 da GRN – Locar o Recurso

Como os tipos de requisitos já foram definidos e os critérios de teste já foram selecionados na Subseção 5.2.1, a descrição da execução dos **PASSOS 1 e 2** da estratégia não são comentados nesta subseção.

Inicialmente, de acordo com o **PASSO 3.1** e **PASSO 3.2**, foram definidas classes de equivalência, comuns a todos os padrões da GRN, relacionadas à consistência e à integridade de dados, conforme apresentado na Tabela 5.6. As classes de equivalência de consistência foram baseadas nos *scripts* em SQL (*Structured Query Language*) que o framework GREN utiliza, durante sua instanciação, para gerar as tabelas do sistema correspondentes às classes dos padrões da GRN, pois o tipo e o tamanho de cada atributo das classes não estavam

especificados na estrutura do padrão, conforme pode ser visto na estrutura do padrão 4, apresentada na Figura 5.2. As classes de equivalência de integridade foram baseadas em algumas regras de integridade de banco de dados relacionais, como comentado na Subseção 5.2.1.

Tabela 5.6: Classes de equivalência de consistência e de integridade

Operação	**Tipo do Requisito**	**Classes Válidas**	**Classes Inválidas**
Inserção, Alteração, Remoção	Consistência	valor inteiro **(1)**	valor não inteiro **(2)**
Inserção, Alteração, Remoção	Consistência	valor entre 1 e 2147483647 **(3)**	valor < 1 **(4)**
			valor > 2147483647 **(5)**
Inserção, Alteração, Remoção	Consistência	valor entre 0 e 2147483647 **(6)**	valor < 0 **(7)**
Inserção, Alteração, Remoção	Consistência e Integridade	valor preenchido **(8)**	valor vazio **(9)**
Inserção, Alteração	Consistência	valor alfanumérico **(10)**	valor não alfanumérico **(11)**
Inserção, Alteração	Consistência	tamanho <= 35 **(12)**	tamanho > 35 **(13)**
Inserção, Alteração	Consistência	tamanho == 1 **(14)**	tamanho != 1 **(15)**
Inserção, Alteração	Consistência	tamanho <= 60 **(16)**	tamanho > 60 **(17)**
Inserção, Alteração	Consistência	tamanho <= 10 **(18)**	tamanho > 10 **(19)**
		tamanho <= 20 **(20)**	tamanho > 20 **(21)**
		tamanho <= 30 **(22)**	tamanho > 30 **(23)**
Inserção, Alteração	Consistência	valor float **(24)**	valor não float **(25)**
Inserção, Alteração	Consistência	valor entre 0.0 e 2147483647.0 **(26)**	valor < 0.0 **(27)**
			valor > 2147483647.0 **(28)**
Inserção, Alteração	Consistência	data <= data do dia do sistema **(29)**	data > data do dia do sistema **(30)**
Inserção, Alteração	Consistência	dia inteiro **(31)**	dia não inteiro **(32)**
Inserção, Alteração	Consistência	mês inteiro **(33)**	mês não inteiro **(34)**
Inserção, Alteração	Consistência	ano inteiro **(35)**	ano não inteiro **(36)**
Inserção, Alteração	Consistência	0 < dia <= quantidade de dias válidos para o mês **(37)**	dia <= 0 ou > quantidade de dias válidos para o mês **(38)**
Inserção, Alteração	Consistência	0 < mês <= 12 **(39)**	mês <= 0 ou > 12 **(40)**
Inserção, Alteração	Consistência	ano >= 0100 **(41)**	ano < 0100 **(42)**
Inserção	Integridade	valor de chave primária não cadastrado **(43)**	valor de chave primária cadastrado **(44)**
Alteração, Remoção	Integridade	valor de chave primária cadastrado **(45)**	valor de chave primária não cadastrado **(46)**
Inserção, Alteração	Integridade	valor de chave estrangeira cadastrado como chave primária em tabela associada **(47)**	valor de chave estrangeira não cadastrado como chave primária em tabela associada **(48)**
Remoção	Integridade	valor sem relacionamentos pendentes (não cadastrado como chave estrangeira em tabela associada) **(49)**	valor com relacionamentos pendentes **(50)**
...	...	...	...

Posteriormente, foram criados os requisitos de teste de consistência e de integridade, considerando cada uma das classes de equivalência criadas, válidas e inválidas, e o critério de teste funcional Análise do Valor Limite.

O **PASSO 5** (Documentar cada requisito de teste criado) foi executado em seguida para documentar os requisitos de consistência e de integridade criados. A documentação de alguns dos requisitos de teste criados está apresentada na Tabela 5.7. As oito primeiras linhas da tabela contém a documentação de alguns dos requisitos de teste de consistência criados e a nona e a décima linhas contém a documentação de alguns dos requisitos de teste de integridade, relacionados ao comportamento das chaves primária e estrangeira durante as operações de manipulação de dados.

Posteriormente, o **PASSO 4** (Elaborar requisitos de teste do negócio) foi executado. Nesse passo, criaram-se classes de equivalência relacionadas ao negócio, cujas regras estão embutidas nas classes de cada padrão da GRN em que a estratégia da abordagem ARTe foi aplicada. A criação das classes de equivalência do negócio foi apoiada pelo conhecimento do engenheiro de software no domínio ao qual a linguagem de padrões pertence. Esse conhecimento pode também ser obtido pelo estudo da própria linguagem. Na Tabela 5.8 ilustram-se algumas das classes de equivalência criadas especificamente para a classe `Locação de Recurso` do padrão 4 da GRN. Essa classe embute algumas das regras do negócio que devem ser tratadas durante a locação de recursos. Por exemplo, um recurso só pode ser locado se estiver disponível. No caso de recursos quantificáveis, é necessário verificar se existe quantidade do recurso suficiente para ser locada. A partir das classes de equivalência do negócio criadas, com o apoio do critério de teste Particionamento de Equivalência e do critério de teste funcional Análise do Valor Limite, requisitos de teste foram definidos.

O **PASSO 5** (Documentar cada requisito de teste criado) foi executado em seguida para documentar os requisitos de teste do negócio criados. Na Tabela 5.9 apresentam-se alguns dos requisitos de teste criados para a classe `Locação de Recurso` do padrão 4 da GRN, a partir das classes de equivalência apresentadas na Tabela 5.8.

Depois de criar e documentar os requisitos de teste, o **PASSO 6** (Derivar e documentar casos de teste) foi executado. Nesse passo, casos de teste foram derivados tomando como base os requisitos de teste de consistência e de integridade, criados nos **PASSOS 3.1** e **3.2**, respectivamente, e os requisitos relacionados ao negócio, criados no **PASSO 4**. Na Tabela 5.10 apresentam-se parcialmente os casos de teste de consistência e de integridade derivados a partir dos requisitos de teste apresentados na Tabela 5.7. Na Tabela 5.11 apresentam-se alguns dos casos de teste criados para a classe `Locação de Recurso` do padrão 4 da GRN, a partir de alguns requisitos de teste definidos na Tabela 5.9.

Tabela 5.7: Documentação parcial dos requisitos de teste de consistência e de integridade dos padrões da GRN

Nr. Req. Teste	Tipo	Operação	Classes de Equiv.	Espec. do Req. de Teste	Condições Válidas	Observação	Retorno Esperado
RT01	Consist.	Inserção, Alteração, Remoção, Consulta	2	atributo não inteiro	atributo inteiro (maior que 0 e menor que 2147483648)	–	falha na verificação da consistência, não permitindo continuar com a operação
RT02	Consist.	Inserção, Alteração, Remoção, Consulta	1,4,8	atributo == 0	atributo valor inteiro (maior que 0 e menor que 2147483648)	–	falha na verificação da consistência, não permitindo continuar com a operação
RT03	Consist.	Inserção, Alteração, Remoção, Consulta	1,3,8	atributo == 1	atributo valor inteiro (maior que 0 e menor que 2147483648)	–	sucesso na verificação, permitindo continuar com a ação
RT04	Consist.	Inserção, Alteração, Remoção, Consulta	1,3,8	atributo == 2147483647	atributo valor inteiro (maior que 0 e menor que 2147483648)	–	sucesso na verificação, permitindo continuar com a ação
RT05	Consist.	Inserção, Alteração, Remoção, Consulta	2,5,8	atributo > 2147483647	atributo valor inteiro (maior que 0 e menor que 2147483648)	–	falha na verificação, não permitindo continuar com a ação
RT06	Consist.	Inserção, Alteração, Remoção, Consulta	1,4,8	atributo < 1	atributo valor inteiro (maior que 0 e menor que 2147483648)	–	falha na verificação, não permitindo continuar com a ação
RT07	Consist.	Inserção, Alteração, Remoção, Consulta	9	atributo vazio	atributo não é vazio	–	falha na verificação, não permitindo continuar com a ação
RT08	Consist.	Inserção, Alteração, Remoção, Consulta	10	atributo não alfanumérico	atributo alfanumérico	–	falha na verificação da consistência, não permitindo continuar com a operação
RT09	Integr.	Inserção	44	registro cadastrado	registro não cadastrado	–	falha na verificação da integridade, não permitindo continuar com a operação
RT010	Integr.	Alteração, Remoção, Consulta	46	registro não cadastrado	registro cadastrado	–	falha na verificação da integridade, não permitindo continuar com a operação
...	...	...	...	...	...	...	...

Tabela 5.8: Classes de equivalência para o requisito de teste do negócio da classe `Locação de Recurso` do padrão 4 da GRN

Operação	**Classe do Padrão**	**Tabela Correspondente**	**Atributo**	**Classes Válidas**	**Classes Inválidas**	**Observação**
Inserção, Alteração	Locação de Recurso	Resource Rental	situação da instância do recurso	situação da instância do recurso disponível **(51)**	situação da instância do recurso não disponível **(52)**	Considerar se não foi utilizado o padrão 11 (ITEMIZAR A TRANSAÇÃO DO RECURSO) e se foi utilizado o sub-padrão "Recurso Instanciável" do padrão 2 (QUANTIFICAR O RECURSO).
Inserção, Alteração	Locação de Recurso	Resource Rental	quantidade	quantidade do recurso >= quantidade do recurso para ser locada **(53)**	quantidade do recurso < quantidade do recurso para ser locada **(54)**	Considerar se não foi utilizado o padrão 11 e se foi utilizado o sub-padrão "Recurso Mensurável" ou "Recurso em Lotes" do padrão 2.
Alteração	Locação de Recurso	Resource Rental	valor da multa	data de devolução > data final **(55)**	data de devolução <= data final **(56)**	Considerar se o participante "Taxa de Multa" do padrão 4 (LOCAR O RECURSO) foi utilizado.
...	...	...	...	...	...	...

Tabela 5.9: Documentação parcial dos requisitos de teste do negócio da classe `Locação de Recurso` do Padrão 4 da GRN

Nr. Req. Teste	Tipo	Padrão	Classe do Padrão	Operação	Tabela SGBD	Classes de Equiv.	Espec. do Req. de Teste	Condições Válidas	Validações Anteriores	Observação	Retorno Esperado
RT11	do Negócio	Padrão 4: Locar o Recurso	Locação de Recurso	Inserção, Alteração	Resource Rental	51	situação da instância do recurso == "disponível"	situação do recurso == "disponível"	todos os requisitos de teste que consistem os atributos da classe Locação de Recurso	–	sucesso na verificação da funcionalidade do negócio, permitindo continuar com a operação
RT12	do Negócio	Padrão 4: Locar o Recurso	Locação de Recurso	Inserção, Alteração	Resource Rental	52	situação da instância do recurso == "locado"	situação do recurso == "disponível"	todos os requisitos de teste que consistem os atributos da classe Locação de Recurso	–	falha na verificação da funcionalidade do negócio, não permitindo continuar com a operação
RT13	do Negócio	Padrão 4: Locar o Recurso	Locação de Recurso	Inserção, Alteração	Resource Rental	53	situação da instância do recurso == ""	situação do recurso == "disponível"	todos os requisitos de teste que consistem os atributos da classe Locação de Recurso	–	falha na verificação da funcionalidade do negócio, não permitindo continuar com a operação

Tabela 5.10: Casos de teste de consistência e de integridade dos padrões da GRN

Nr. Caso Teste	**Nr. Req. Teste**	**Operação**	**Validações Anteriores**	**Dado de Entrada**	**Saída Esperada**
CT01	RT01	Inserção, Alteração, Remoção, Consulta	–	atributo := "A"	falha na verificação da consistência, não permitindo continuar com a operação
CT02	RT02	Inserção, Alteração, Remoção, Consulta	–	atributo := 0	falha na verificação da consistência, não permitindo continuar com a operação
CT3	RT03	Inserção, Alteração, Remoção, Consulta	–	atributo := 1	sucesso na verificação, permitindo continuar com a ação
CT4	RT04	Inserção, Alteração, Remoção, Consulta	–	atributo := 2147483647	sucesso na verificação, permitindo continuar com a ação
CT5	RT05	Inserção, Alteração, Remoção, Consulta	–	atributo := 2147483648	falha na verificação, não permitindo continuar com a ação
CT6	RT06	Inserção, Alteração, Remoção, Consulta	–	atributo := -1	falha na verificação, não permitindo continuar com a ação
CT7	RT07	Inserção, Alteração, Remoção, Consulta	–	atributo := null	falha na verificação, não permitindo continuar com a ação
CT8	RT8	Inserção, Alteração, Remoção, Consulta	–	atributo := 99	falha na verificação, não permitindo continuar com a ação
CT9	RT09	Inserção	CT01 a CT08	atributo := 1 (já cadastrado)	falha na verificação da integridade, não permitindo continuar com a operação
CT10	RT10	Alteração, Remoção, Consulta	CT01 a CT08	atributo:= 2 (não cadastrado)	falha na verificação da integridade, não permitindo continuar com a operação
...	...	...	...	...	...

Tabela 5.11: Casos de teste do negócio do padrão `Locação de Recurso` da GRN

Nr. Caso Teste	**Nr. Req. Teste**	**Operação**	**Padrão**	**Classe do Padrão**	**Validações Anteriores**	**Dado de Entrada**	**Saída Esperada**
CT11	RT11	Inserção, Alteração	Padrão 4: Locar o Recurso	Locação de Recurso	CT1 a CT11, ...	situação do recurso := "1" (esse valor é considerado pelo framework GREN como recurso disponível)	sucesso na verificação da funcionalidade do negócio, permitindo continuar com a operação
CT12	RT12	Inserção, Alteração	Padrão 4: Locar o Recurso	Locação de Recurso	CT1 a CT11, ...	situação do recurso := "2" (esse valor é considerado pelo framework GREN como recurso locado)	falha na verificação da funcionalidade do negócio, não permitindo continuar com a operação
CT13	RT13	Inserção, Alteração	Padrão 4: Locar o Recurso	Locação de Recurso	CT1 a CT11, ...	situação do recurso := ""	falha na verificação da funcionalidade do negócio, não permitindo continuar com a operação
...	...	...	...	...	...	...	...

No **PASSO 7** (Mapear casos de teste comuns) realizou-se um mapeamento dos casos de teste de integridade e de consistência, que são comuns a todos os padrões da GRN, para os respectivos atributos das classes dos padrões da GRN considerados durante a aplicação da estratégia da abordagem ARTe. Para realizar esse mapeamento, as características de cada atributo (tipo, tamanho, envolvimento em regras de integridade) foram observadas. A partir disso, identificou-se as classes de equivalência de integridade e de consistência relacionadas a tais características e, conseqüentemente, os requisitos e os casos de teste. Na Tabela 5.12 ilustra-se o mapeamento de casos de teste para alguns atributos da classe `Locação de Recurso` do padrão 4 da GRN.

No **PASSO 8** (Disponibilizar os recursos de teste), os recursos de teste do negócio e o mapeamento dos casos de teste de integridade e de consistência para cada classe de cada padrão, realizado no passo anterior, foram disponibilizados na seção *Informações VV&T* do respectivo padrão. Por outro lado, os recursos de teste comuns a todos os padrões foram disponibilizados na seção *Informações VV&T* da linguagem de padrões GRN. Por exemplo, as classes de equivalência do negócio apresentadas na Tabela 5.8, os requisitos de teste apresentados na Tabela 5.9 e os casos de teste apresentados na Tabela 5.11 são disponibilizados na seção *Informações VV&T* do padrão 4 da GRN, enquanto as classes de equivalência de consistência e de integridade ilustradas na Tabela 5.6, os requisitos e casos de teste ilustrados nas Tabelas 5.7 e 5.10, respectivamente, foram disponibilizados na seção *Informações VV&T* da GRN.

Tabela 5.12: Mapeamento dos casos de teste

Padrão	**Classe do padrão**	**Tabela SGBD**	**Atributo**	**Nr. Caso Teste**	**Validações anteriores**
Padrão 4: Locar o Recurso	Locação de Recurso	ResourceRental	número	CT1 a CT8, CT10	–
Padrão 4: Locar o Recurso	Locação de Recurso	ResourceRental	observação	CT9,...	CT1 a CT8
Padrão 4: Locar o Recurso	Locação de Recurso	ResourceRental	situação	CT1,...	CT2 a CT8
...	...	...	...	...	...

5.3.2 Uso das Diretrizes para Reúso dos Recursos de Teste - Um Estudo de Caso

Os resultados do estudo de caso, apresentados nesta subseção, são apenas indícios de que o tempo de reengenharia é menor quando se reutiliza recursos de teste agregados aos padrões da GRN; no entanto, outros estudos controlados devem ser conduzidos para comprovar essa hipótese.

Definição do estudo de caso

Objeto de Estudo: Abordagem de Reúso de Teste ARTe.

Propósito: Estudo de caso para avaliar o reúso dos requisitos de teste, criados com o apoio da abordagem ARTe, na reengenharia de sistemas.

Foco qualitativo: Redução do tempo da reengenharia de sistemas procedimentais para o paradigma OO com o apoio do PARFAIT utilizando a abordagem ARTe.

Perspectiva: A perspectiva é em relação a engenheiros de software interessados na reengenharia de sistemas no domínio de Gestão de Recursos de Negócios.

Contexto: O estudo de caso foi realizado por um aluno de iniciação científica, que cursava o segundo semestre do segundo ano do curso de Bacharelado em Informática no ICMC-USP, e teve como material básico a documentação do PARFAIT, a linguagem de padrões GRN, a documentação das classes de equivalência e requisitos de teste associados aos padrões dessa linguagem; a documentação da ferramenta de instanciação GREN-Wizard e um material de treinamento da linguagem Smalltalk, do SGBD MySQL e dos critérios de teste Particionamento de Equivalência e Análise do Valor Limite; e também das diretrizes de reúso da abordagem ARTe. O estudo é classificado de objeto único (ou seja, um objeto e um participante) e foi iniciado no mês de agosto de 2004. O período inicialmente estimado para a conclusão do estudo de caso foi de cento e setenta e três dias (trabalhando quatro horas/dia).

Planejamento do estudo de caso

Seleção do Contexto: O estudo de caso foi conduzido de forma independente, sendo que o aluno pôde estipular seus próprios horários para a condução do estudo de caso, desde que o controle dos horários de início e fim das atividades do PARFAIT fosse anotado em uma planilha fornecida. O aluno conhecia o SGBD MySQL e não tinha conhecimento da linguagem de padrões GRN, do framework GREN e da linguagem Smalltalk; no entanto tinha conhecimento sobre reengenharia de software obtido a partir da leitura de livros e artigos técnicos. O sistema legado que foi utilizado no estudo de caso é de pequeno porte, foi desenvolvido em Clipper e estava em uso no momento da condução do estudo. Esse sistema legado controla o empréstimo e a devolução de livros de uma biblioteca universitária, possui aproximadamente 6 KLOC e foi utilizado também no estudo de caso apresentado na Seção 3.5 do Capítulo 3. Este estudo é válido em um contexto específico do domínio de Engenharia de Software.

Formulação das Hipóteses

Hipótese nula:

- H01: O tempo de reengenharia quando se usa o processo PARFAIT com a abordagem de reúso de teste ARTe é maior do que quando não se usa essa abordagem.

Hipótese alternativa:

- HA1: O tempo de reengenharia quando se usa o processo PARFAIT com a abordagem de reúso de teste ARTe é menor do que quando não se usa essa abordagem.

Seleção das variáveis

Variáveis independentes: A *experiência* do participante no paradigma OO, na linguagem de padrões GRN, nas diretrizes de reúso da abordagem ARTe, no framework GREN e no domínio de Gestão de Recursos de Negócios; o *nível de escolaridade* e a *especialidade* do participante nas áreas de computação são fatores que influenciam o uso do PARFAIT e das diretrizes de reúso da abordagem ARTe.

Variáveis dependentes: quantidade de casos de teste criados a partir de classes de equivalência reutilizadas, quantidade de casos de teste criados a partir do nada, quantidade de casos de teste criados a partir de requisitos adaptados, quantidade de casos de teste criados a partir de requisitos de teste reutilizados, quantidade total de casos de teste.

Seleção dos indivíduos: A técnica de escolha foi a amostragem por conveniência, uma vez que o participante era um aluno voluntário de iniciação científica.

Projeto do estudo de caso: O estudo de caso foi realizado por um participante em apenas um projeto de reengenharia.

Instrumentação: Documento do processo PARFAIT (Cagnin et al., 2004g); linguagem de padrões GRN (Braga et al., 1999); documento dos requisitos de teste funcional e das classes de equivalência dos padrões 1 a 4 da GRN; diretrizes de reúso da abordagem ARTe; formulário para coleta de dados do estudo de caso; manual da ferramenta de instanciação GREN-Wizard (Braga, 2002a) e da ferramenta *GREN-WizardVersionControl* (Capítulo 6); documento de treinamento da linguagem de programação Smalltalk, do SGBD MySQL e dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite; sistema legado executável, framework GREN, ferramenta de instanciação GREN-Wizard e ferramenta *GREN-WizardVersionControl*.

Avaliação da validade: *Validade de conclusão:* alguns dados coletados são subjetivos e dependem da qualificação do engenheiro de software que está utilizando o processo; *Validade interna:* o estudo de caso foi realizado pelo aluno sem restrição de dia, horário e local, portanto ele pôde aplicá-lo no momento que achou mais adequado. *Validade de construção:* como o estudo de caso foi realizado somente com um participante e com um objeto, muito provavelmente não dará a visão completa do que se pretende provar; *Validade externa:* esse estudo de caso foi conduzido no meio acadêmico, mas em um sistema legado real.

Operação do Estudo de Caso

Execução: O estudo de caso foi conduzido em duas etapas, similarmente ao estudo de caso apresentado na Seção 3.5 do Capítulo 3. A primeira etapa consistiu no treinamento

e no estudo das técnicas necessárias para o início do estudo de caso: 20 horas para o treinamento e aprendizado da documentação do PARFAIT, 8 horas para o treinamento e aprendizado da linguagem de padrões GRN, aproximadamente 60 horas para o treinamento e aprendizado da ferramenta de instanciação GREN-Wizard, da linguagem de programação Smalltalk e do SGBD MySQL, 2 horas para o treinamento dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite e 0,5 hora para o treinamento das diretrizes de reúso da abordagem ARTe. Não houve necessidade de treinamento da ferramenta *GREN-WizardVersionControl* pois, como comentado no Capítulo 6, o projeto de sua interface foi baseado em uma das telas do ambiente de desenvolvimento do VisualWorks (Cincom, 2003), com a qual o aluno estava familiarizado devido ao treinamento obtido em Smalltalk.

A segunda etapa consistiu na aplicação do PARFAIT durante a reengenharia do sistema de controle de biblioteca, com o apoio das diretrizes de reúso de teste da abordagem ARTe. Durante essa etapa o aluno coletou os seguintes dados: seqüência da execução das atividades do PARFAIT, tempo gasto em cada atividade e na inspeção dos artefatos nela produzidos, tempo total gasto nos marcos de referência, bem como o número de erros encontrados durante as inspeções de cada atividade, conforme apresentado na Tabela 5.13.

A totalização do tempo gasto nos quatro marcos de referência (três horas) está computada no tempo total da reengenharia, apresentado na Tabela 5.13. Nessa tabela apresenta-se também o tempo estimado na atividade "Elaborar o planejamento do projeto de reengenharia" para executar cada atividade. O tempo das atividades com nome em itálico e a quantidade de erros encontrados nas inspeções dessas atividades não são computados no total, pois essas atividades foram conduzidas antes do planejamento. Estimou-se também um tempo de 1 hora para executar cada marco de referência do PARFAIT e em torno de 66 horas para realizar as inspeções de garantia de qualidade nos artefatos produzidos.

Ressalta-se que das 253:10 horas despendidas na atividade "Desenvolver o diagrama de casos de uso e documentar os casos de teste", 15:10 horas não estão relacionadas a atividades VV&T, mas sim ao entendimento da funcionalidade do sistema legado e à construção do diagrama de casos de uso.

Outros dados foram coletados durante a condução do estudo de caso e são apresentados na Tabela 5.14. Esses dados apoiaram a análise das hipóteses definidas. De acordo com os dados apresentados nessa tabela, destaca-se que em torno de 81% dos casos de teste criados foram baseados no reúso de requisitos de teste disponíveis nos padrões, colaborando para reduzir o tempo gasto com atividades VV&T na reengenharia de sistemas.

Tabela 5.13: Coleta de dados do estudo de caso para avaliar o reúso de teste

Seqüência das atividades executadas	**Atividades do PARFAIT**	**Tempo estimado por atividade**	**Tempo gasto por atividade**	**Tempo estimado/ gasto na inspeção**	**Qtde erros encontrados na inspeção**
1	*Familiarizar-se com o domínio do framework*	–	3:00	-/0:30	0
2	*Observar o domínio do sistema legado em relação ao do framework*	–	0:20	–/0:10	0
3	*Confrontar as características não funcionais do framework com as do sistema legado*	–	1:00	–/0:20	2
4	*Elaborar o planejamento do projeto de reengenharia*	–	0:40	–/0:30	0
5,18	Desenvolver o diagrama de casos de uso e documentar os casos de teste	504:00	253:10	16:00/32:45	15
6,9,13,17	Desenvolver o diagrama de classes do sistema alvo	15:50	17:20	1:30/1:15	1
7,10,14,19	Documentar as modificações realizadas no diagrama de classes	9:00	3:20	1:00/0:35	0
8	Documentar as regras do negócio do sistema	9:00	1:30	1:00/0:10	0
11	Criar o sistema alvo no paradigma orientado a objetos	9:00	0:20	1:00/0:10	0
12,16,21	Executar os casos de teste no sistema alvo	34:30	08:30	3:30/0:40	0
15,20	Adaptar o sistema alvo	58:00	18:00	4:00/1:40	0
não realizada	Elaborar o manual do usuário do sistema	–	–	–/–	–
22,24	Converter a base de dados do sistema legado	13:00	9:30	8:00/1:15	0
23,25	Testar o sistema alvo	30:00	04:00	30:00/0:20	0
não realizada	Treinar os usuários finais	–	–	–	–
	Marcos de referência		3:00		
	Total	682:20	**323:40**	**66:00/73:15**	**16**

Tabela 5.14: Outros dados coletados durante o estudo de caso

Dado coletado	Valor
Quantidade de casos de teste criados a partir de classes de equivalência reutilizadas:	34 (4%)
Quantidade de casos de teste criados a partir do nada (novas classes de equivalência):	43 (5%)
Quantidade de casos de teste criados a partir dos requisitos reutilizados e adaptados:	80 (9,4%)
Quantidade de casos de teste criados a partir de requisitos de teste reutilizados:	695 (81,6%)
Quantidade total de casos de teste:	852 (100%)

O aluno identificou as regras do negócio no início da reengenharia, juntamente com o usuário do sistema legado, durante a atividade "Desenvolver o diagrama de casos de uso e documentar os casos de teste". Dessa forma, não houve necessidade de instanciar o framework novamente.

Análise e Interpretação dos Resultados do Estudo de Caso

Em geral, de acordo com a Tabela 5.13, pode-se observar que o tempo gasto com as inspeções foram menores do que o tempo estimado na atividade "Elaborar o planejamento do projeto de reengenharia", exceto na inspeção dos artefatos da atividade "Desenvolver o diagrama de casos de uso e documentar os casos de teste", principalmente o de documentação dos casos de teste, que consumiu maior tempo de inspeção. De acordo com o aluno de IC, a razão disso foi porque ele nunca teve experiência com teste de software, tendo que algumas vezes corrigir a documentação de alguns casos de teste.

Ressalta-se que a criação do artefato "Planejamento da Reengenharia" foi baseada em um projeto anterior de reengenharia, para o mesmo sistema, feito por uma outra aluna de iniciação científica cujos resultados estão apresentados na Seção 3.5 do Capítulo 3.

Neste estudo de caso, do tempo total gasto na atividade "Desenvolver o diagrama de casos de uso e elaborar os casos de teste", o aluno gastou 15:10 horas para elaborar o diagrama de casos de uso e entender o sistema legado. Assim, o tempo total gasto com atividades de VV&T neste estudo de caso foi de 250:40 horas, ou seja, 238:10 horas na atividade "Desenvolver o diagrama de casos de uso e elaborar os casos de teste" para elaborar os casos de teste, 8:30 horas na atividade "Executar os casos de teste no sistema alvo" e 4 horas na atividade "Testar o sistema alvo". A partir desses valores, observou-se que em torno de 77% de todo o tempo da reengenharia foi gasto principalmente para elaborar os artefatos relacionados a atividades de VV&T.

Salienta-se que houve uma redução de tempo em torno de 57% na criação dos artefatos referentes a VV&T na atividade "Desenvolver um diagrama de casos de uso e documentar os casos de teste", com o reúso de requisitos de teste associados aos padrões da GRN e com a aplicação das diretrizes de reúso da abordagem ARTe, em relação aos resultados do estudo

de caso da Seção 3.5 do Capítulo 3. Essa mesma porcentagem de redução foi mantida em relação ao tempo total gasto com atividades VV&T durante todo o estudo de caso, conforme apresentado na Figura 5.3. Conseqüentemente, o tempo total da reengenharia sofreu também uma redução significativa de 52%. Esses resultados foram alcançados pois a maioria dos casos de teste foram criados com o apoio de requisitos reutilizados completamente, como pode ser visto na quarta linha da Tabela 5.14. Acredita-se que a redução do tempo da atividade "Desenvolver um diagrama de casos de uso e documentar os casos de teste" pode diminuir ainda mais se, além dos requisitos, os casos de teste disponíveis, associados aos padrões da linguagem, também forem reutilizados. Para comprovar isso, é necessário realizar outros estudos de caso controlados e com um número significativo de participantes, a fim de rejeitar ou não estatisticamente a hipótese nula.

Outro ponto observado foi que a porcentagem do tempo total da reengenharia gasto com atividades de VV&T sofreu pouca alteração (77%) em relação ao estudo de caso anterior (Seção 3.5, Capítulo 3), que foi de 86%. Isso pode ser justificado porque nas demais atividades do PARFAIT, não relacionadas a VV&T, o tempo gasto foi semelhante nos dois estudos de caso. Outra informação obtida neste estudo de caso foi que houve um aumento significativo na quantidade de casos de teste, em torno de 96%. Uma razão para isso é que os casos de teste do estudo de caso anterior foram criados a partir da funcionalidade do sistema legado, enquanto que neste estudo de caso, os casos de teste foram criados a partir do reúso dos recursos de teste do domínio. Isso pode ser uma vantagem, pois todo comportamento da funcionalidade do domínio está sendo considerado durante a atividade de teste, podendo levar a uma maior cobertura de teste. Ressalta-se que nenhum aspecto relacionado a cobertura de teste foi observado em ambos estudos de caso.

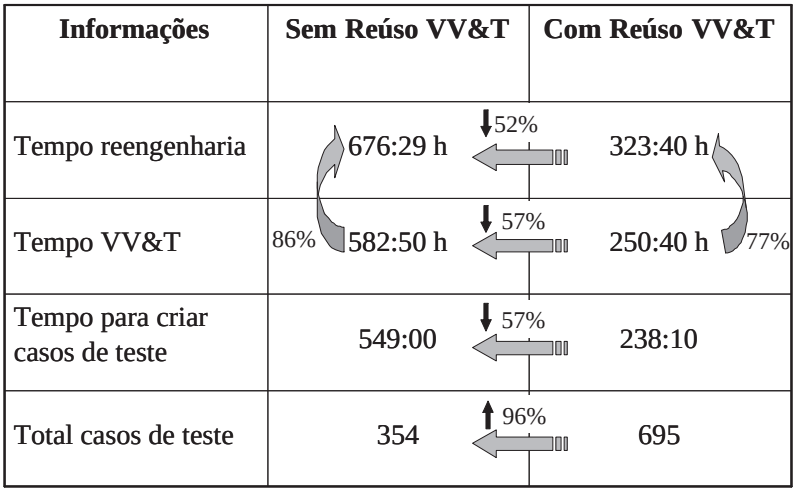


Figura 5.3: Comparação dos dados de dois estudos de caso

Discussão do Estudo de Caso

Além da análise quantitativa dos resultados, apresentada nas Tabelas 5.13 e 5.14, uma análise qualitativa foi realizada com base no questionário de *feedback*, respondido pelo aluno que realizou o estudo de caso. Segundo ele, o uso do framework e da linguagem de padrões, bem como dos requisitos de teste, agilizaram o desenvolvimento do projeto de reengenharia. Além disso, o ponto de maior complexidade do projeto foi a criação dos casos de teste a partir dos recursos de teste disponíveis. Esse passo demandou quantidade de tempo considerável, tanto para sua execução, quanto para o entendimento do reúso correto dos requisitos de teste e das classes de equivalência, associados aos padrões da GRN. No entanto, o aluno afirma que o reúso dos recursos de teste é um ponto positivo do ARA, pois se eles não estivessem disponíveis, a criação dos casos de teste seria mais difícil e demorada. Além disso, a qualidade dos casos de teste e, conseqüentemente, do sistema alvo poderia ser afetada. Apesar de ter tido certa dificuldade no reúso dos recursos de teste no início, o aluno afirmou que os requisitos ajudaram a poupar esforço e tempo na reengenharia. O aluno sugeriu mais tempo de treinamento nas diretrizes de reúso da abordagem ARTe, considerando inclusive exercícios de fixação. Com isso, pode ser possível diminuir o tempo gasto no reúso dos recursos de teste. De acordo com o aluno, o uso da linguagem de padrões GRN apoiou tanto no entendimento do sistema, quanto na elaboração do diagrama de classes, e o framework GREN minimizou o esforço de programação do sistema alvo, uma vez que foi necessário somente refinar o código fonte gerado a partir do framework para adequá-lo à funcionalidade do sistema legado.

5.4 Considerações Finais

O uso completo da abordagem ARTe foi apresentado na Seção 5.3. Primeiramente foi relatado o uso da estratégia de definição de recursos de teste funcional em um dos padrões da GRN. Em seguida, foi descrito um estudo de caso de reengenharia em que foram reutilizados, durante as atividades de VV&T, os requisitos de teste definidos. Para apoiar esse reúso, as diretrizes de reúso de teste da abordagem foram utilizadas. A partir desse estudo de caso observou-se uma redução em torno de 57% do tempo gasto com as atividades de VV&T em relação a um outro estudo realizado sem o reúso de requisitos de teste (Capítulo 3, Seção 3.5). Outros estudos de caso devem ser conduzidos para avaliar a redução do tempo de reengenharia quando se reutiliza não somente os requisitos mas também os casos de teste. Ressalta-se que a abordagem ARTe deve ser aplicada em outras linguagens de padrões, com outras particularidades para verificar o quanto ela é flexível. Além disso, o uso dessa abordagem por outras pessoas em outros contextos colaborará para o seu refinamento e evolução.

Como padrões de LPAs fornecem conhecimento sobre um determinado domínio, na forma de soluções de problemas de análise, ao invés de fornecer informação do código fonte, nesta

tese foram tratados apenas requisitos de teste funcional. Ressalta-se a importância de definir outras estratégias para criar requisitos de teste, baseados em critérios estruturais e em outras técnicas específicas para testar frameworks (Dallal e Sorenson, 2002; Jeon et al., 2002; Tsai et al., 1999), para associá-los às classes de frameworks baseados em LPAs, como é o caso do GREN. Dessa forma, poder-se-á obter uma cobertura maior dos testes e, conseqüentemente, a criação de um produto com maior qualidade, tanto resultante do desenvolvimento quanto da reengenharia.

Outro ponto observado é que a abordagem definida colabora para a prática ágil do teste antes da escrita do código, pois fornece recursos de teste associados à especificação do software, no caso, por meio dos padrões da LPA.

No próximo capítulo apresentam-se a ferramenta *GREN-WizardVersionControl*, que foi criada especialmente para facilitar o alcance da iteratividade do arcabouço ARA, e um exemplo de uso utilizando essa ferramenta. Essa ferramenta está disponível neste arcabouço e pode ser utilizada tanto na reengenharia quanto no desenvolvimento incremental baseado em framework.

Capítulo

6

GREN-WizardVersionControl: Uma Ferramenta de Apoio ao Controle de Versões no PARFAIT

6.1 Considerações Iniciais

Neste capítulo apresenta-se uma ferramenta, desenvolvida nesta tese, de controle das versões das aplicações criadas a partir da instanciação de um determinado framework. A necessidade de desenvolver essa ferramenta, denominada *GREN-WizardVersionControl*, foi evidenciada principalmente durante um estudo de caso de reengenharia utilizando o PARFAIT e, conseqüentemente o ARA, com apoio computacional do framework GREN (Seção 3.5 do Capítulo 3). Durante tal estudo de caso observou-se que frameworks de aplicação não apóiam a reengenharia e o desenvolvimento quando se usa um modelo de processo incremental e iterativo, o qual produz o software após constantes iterações. Dessa forma, não é possível gerar uma nova versão de uma determinada aplicação com outros requisitos herdados do framework, sem perder as modificações realizadas manualmente no código fonte, em versões anteriores. Por isso, é necessário considerar durante todo o processo de desenvolvimento/reengenharia de software o controle de versão, que é uma das atividades do Gerenciamento de Controle de Configuração do Software, e é um elemento importante para garantir a qualidade do software (Pressman, 2005). Esta ferramenta foi associado ao arcabouço ARA para aumentar sua efetividade na reengenharia de software.

Ressalta-se que a problemática do controle de versão no contexto de frameworks de aplicação é mais grave do que no desenvolvimento de software convencional, pois é necessário controlar tanto as versões do framework quanto as das aplicações criadas por meio de sua instanciação. *GREN-WizardVersionControl* preocupa-se apenas com o controle de versão das aplicações criadas a partir do framework GREN.

Além de evidenciar a necessidade de criar a ferramenta *GREN-WizardVersionControl*, observou-se também a necessidade de evoluir a ferramenta de instanciação GREN-Wizard, para que ela pudesse incorporar, durante a criação da nova versão de uma determinada aplicação, as alterações realizadas no código fonte das suas versões anteriores, gerenciadas pela ferramenta *GREN-WizardVersionControl*.

Foi encontrada na literatura uma ferramenta, proposta por Tourwé (Tourwé, 2002), com objetivo diferente da ferramenta apresentada neste capítulo, que controla as mudanças causadas na evolução de frameworks e das aplicações criadas a partir deles. Tal ferramenta avalia o impacto que as mudanças podem ter tanto na hierarquia de classes do framework quanto na das aplicações geradas a partir dele.

Este capítulo está organizado da seguinte forma: na Seção 6.2 são apresentados a funcionalidade suportada pela ferramenta *GREN-WizardVersionControl*, uma discussão sobre as suas limitações, o modelo conceitual criado para representar os dados referentes ao controle de versão, bem com as características de implementação e de arquitetura da ferramenta. Na Seção 6.3 é apresentada a evolução da ferramenta GREN-Wizard que foi feita para integrá-la com a ferramenta *GREN-WizardVersionControl*. Na Seção 6.4 é apresentado um exemplo de uso da ferramenta desenvolvida visando ilustrar suas principais características. Na Seção 6.5 são apresentadas as conclusões parciais do trabalho, enfocando a ferramenta de controle de versão criada.

6.2 Ferramenta GREN-WizardVersionControl

Como não foi possível garantir que o código fonte das versões das aplicações geradas a partir da instanciação do framework GREN estivesse na mesma ordem, o uso de ferramentas existentes de controle de versão (por exemplo, CVS (Concurrent Versions System, 2004)) para realizar o *merge* das versões tornou-se difícil. Além disso, com essas ferramentas não foi possível controlar o conflito do código fonte em tempo de instanciação do framework e nem realizar a manutenção da base de dados da nova versão da aplicação. Isso motivou o desenvolvimento da ferramenta *GREN-WizardVersionControl* (Cagnin et al., 2004b,e). Essa ferramenta apóia a realização da tarefa controle de versão da atividade do Gerenciamento de Configuração de Software (Pressman, 2005; Sommerville, 2000), colaborando para garantir a qualidade do produto resultante tanto do desenvolvimento quanto da reengenharia.

Salienta-se que a ferramenta desenvolvida possui algumas limitações, citadas a seguir:

- é específica a um único framework;
- não fornece controle de *baselines*;
- não fornece controle de *releases*, ou seja, não há acompanhamento dos *releases* das versões do sistema entregues aos usuários;
- não fornece controle de sincronização que garante que mudanças que ocorrem em paralelo, feitas por duas ou mais pessoas diferentes, não sobrescrevam umas as outras. Isso pode dificultar a execução das práticas ágeis "propriedade coletiva do código" e "integração contínua" (Beck, 2000);
- não permite selecionar uma determinada versão da aplicação para ser considerada durante a subseqüente instanciação do framework; considera somente a última versão da aplicação que foi gerada;
- não permite que o engenheiro de aplicação tenha acesso, quando necessário, a diferentes versões de uma determinada aplicação; e
- durante a decisão de conflito, apresenta somente o conteúdo mais atual do método da aplicação que foi modificado, ou seja, não fornece ao engenheiro de aplicação a possibilidade de selecionar uma das versões anteriores daquele método para ser considerada na instanciação que está em execução.

A ferramenta *GREN-WizardVersionControl* deve ser utilizada sempre que houver necessidade de modificar o código fonte de qualquer aplicação, gerada a partir da instanciação do framework GREN. Para isso, possui uma única tela (Figura 6.1), cujo projeto de interface foi baseado em uma das telas do ambiente de desenvolvimento do VisualWorks (Cincom, 2003), a fim de facilitar o uso da ferramenta e possibilitar que o engenheiro de aplicação a utilize como parte da execução de seu trabalho.

Todas as modificações realizadas no código fonte da aplicação, por meio da ferramenta *GREN-WizardVersionControl*, são armazenadas em uma base de dados e são posteriormente consultadas durante a próxima instanciação do framework para a mesma aplicação, a fim de serem incorporadas na versão sendo gerada.

Por meio da *GREN-WizardVersionControl* é possível adicionar atributos básicos (`inteiro, string, float, date`), multivalorado e enumerado; alterar somente atributos básicos e não herdados do framework; adicionar protocolo ou categorias de métodos [1]; e adicionar/alterar classes e métodos.

A adição de atributos, além de ser armazenada na base de dados da *GREN-WizardVersionControl* é também armazenada na base de dados da ferramenta

[1]No ambiente de desenvolvimento VisualWorks, métodos da classe são organizados em protocolos (ou categorias) que são agrupamentos de métodos semanticamente relacionados.

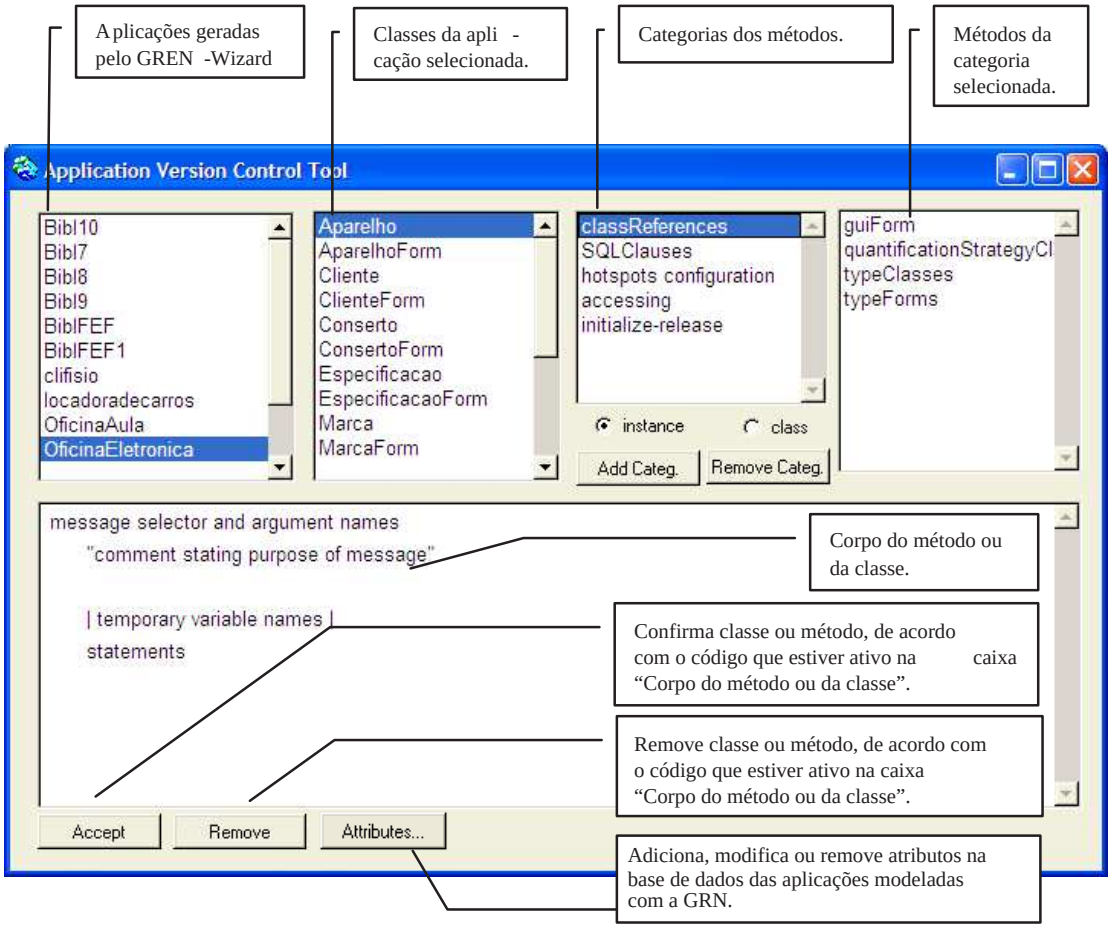


Figura 6.1: Tela principal da ferramenta *GREN-WizardVersionControl*

GREN-Wizard, que mantém um registro das aplicações modeladas com a linguagem de padrões GRN. Para isso, é necessário selecionar a classe desejada para criar os atributos e clicar no botão *Attributes*. Uma tela, similar àquela da ferramenta GREN-Wizard, aparece para a especificação dos novos atributos (nome, tipo e tamanho), conforme apresentado na Figura 6.2. Na próxima instanciação do framework com o apoio da GREN-Wizard, os métodos que tratam do encapsulamento dos dados dos novos atributos especificados são criados automaticamente na nova versão do código fonte da aplicação. A alteração de atributos também é permitida, mas somente para aqueles não herdados do framework.

A ferramenta *GREN-WizardVersionControl* permite a remoção de classes, atributos, categorias de métodos e métodos somente para classes, atributos, categoria de métodos e métodos não herdados do framework, respectivamente. As restrições de remoção foram estabelecidas para garantir que os elementos herdados do framework (por exemplo, classes, atributos, métodos, entre outros) não sejam removidos acidentalmente ou propositadamente do código fonte da aplicação criada a partir do framework. Só é permitido que o engenheiro de aplicação sobreponha ou modifique os métodos herdados, visando garantir que nenhum *hot spot* seja danificado, e que a estrutura da aplicação permaneça correta e funcionando adequadamente.

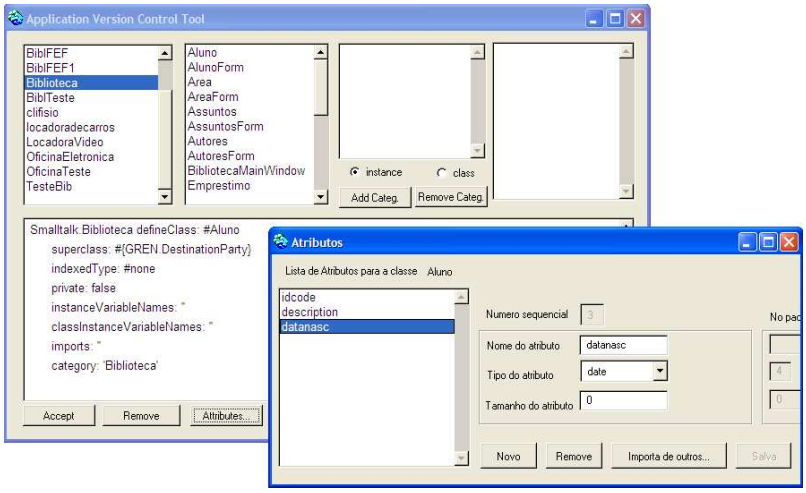


Figura 6.2: Tela que apóia a inserção de novos atributos

6.2.1 Modelo Conceitual

O diagrama de classes apresentado na Figura 6.3 contém as classes do modelo da ferramenta *GREN-WizardVersionControl* e a classe que implementa a interface dessa ferramenta (ou seja, `GrenWizardVersionControl`).

As classes `ClassVersionControl`, `ClassAttributeVersionControl`, `ClassProtocol-VersionControl` e `ClassMethodVersionControl` contêm informações relevantes sobre: a classe, o atributo, o protocolo ou categoria do método e o método, respectivamente, que estão sendo atualizados. Como essas classes contém algumas informações em comum, elas herdam da classe `MetaVersionControl`.

A classe `GREN-WizardVersionControl` faz todo o controle dos elementos do código fonte da aplicação (classe, atributo, método e protocolo de método) que estão sendo modificados e contém uma interface semelhante ao *System Browser*, que é um dos ambientes de programação do VisualWorks (Cincom, 2003).

As classes `ClassVersionControl`, `ClassAttributeVersionControl`, `ClassProtocol-VersionControl`, `ClassMethodVersionControl` foram mapeadas para o SGBD MySQL (MySQL, 2003) e armazenam os elementos do código fonte de cada aplicação que foram modificados, ou seja, cada *item de configuração de software*. O mapeamento foi feito da seguinte forma: a classe `MetaVersionControl` não foi criada como tabela no SGBD, no entanto, seus atributos foram mapeados para cada tabela correspondente às classes `ClassVersionControl`, `ClassAttributeVersionControl`, `ClassProtocolVersionControl` e `ClassMethodVersionControl`.

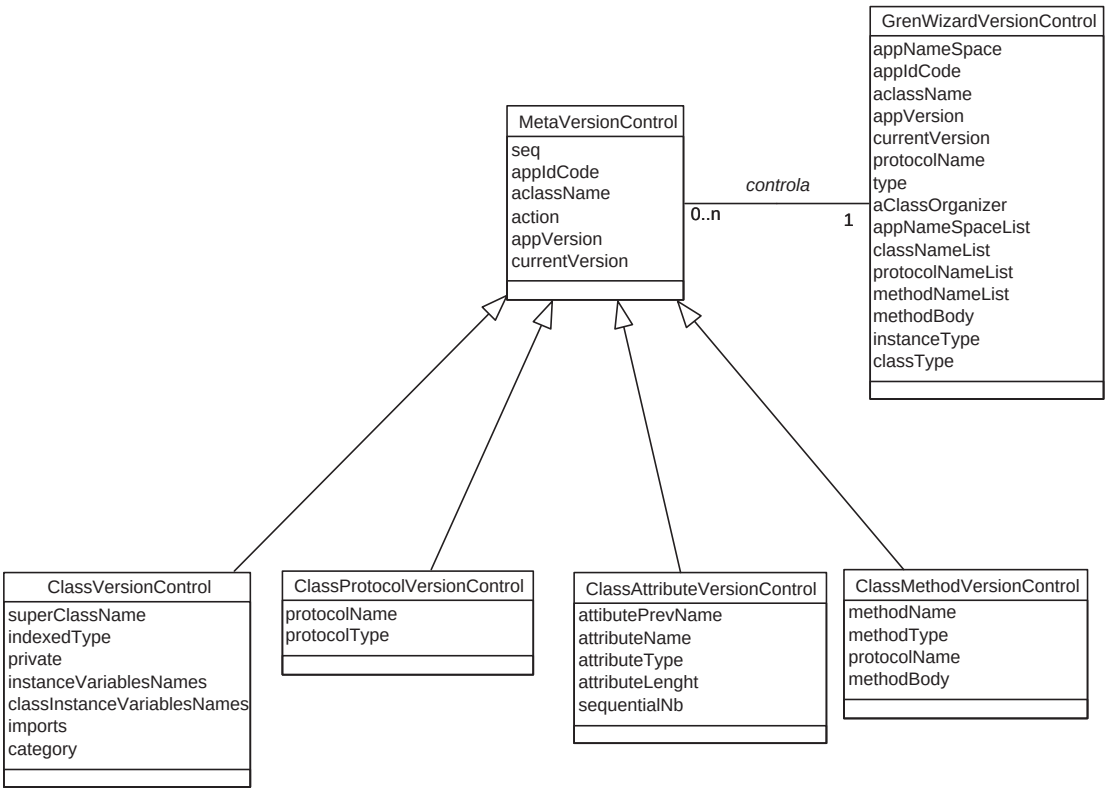


Figura 6.3: Modelo conceitual da ferramenta *GREN-WizardVersionControl*

6.2.2 Arquitetura e Implementação

A arquitetura da ferramenta desenvolvida foi baseada na arquitetura do framework GREN e da ferramenta de instanciação GREN-Wizard (Braga, 2003; Braga e Masiero, 2003) para que as ferramentas pudessem comunicar entre si sem haver necessidade de criar uma camada de integração para isso.

Como já mencionado na Seção 2.4.2.1 do Capítulo 2, a arquitetura do GREN foi projetada em três camadas: a de persistência, a de negócios e a de interface gráfica com o usuário (GUI), conforme apresentado na Figura 6.4. A camada GREN-Wizard, que gera o código da aplicação a partir da sua especificação baseada nos padrões da GRN, comunica-se com a camada *GREN-WizardVersionControl*. Isso é feito para que a informação das modificações feitas nos elementos do código fonte de uma determinada aplicação seja considerada na próxima instanciação do framework.

Na Figura 6.5 apresenta-se a arquitetura das ferramentas GREN-Wizard e *GREN-WizardVersionControl*. Como comentado no Capítulo 2, dois atores interagem diretamente com a ferramenta GREN-Wizard: o engenheiro do domínio, responsável pela construção dessa ferramenta e o engenheiro de aplicações, que a utiliza para especificar aplicações por meio de uma interface gráfica com o usuário (GUI). O usuário final executa o código da aplicação gerado pela ferramenta GREN-Wizard.

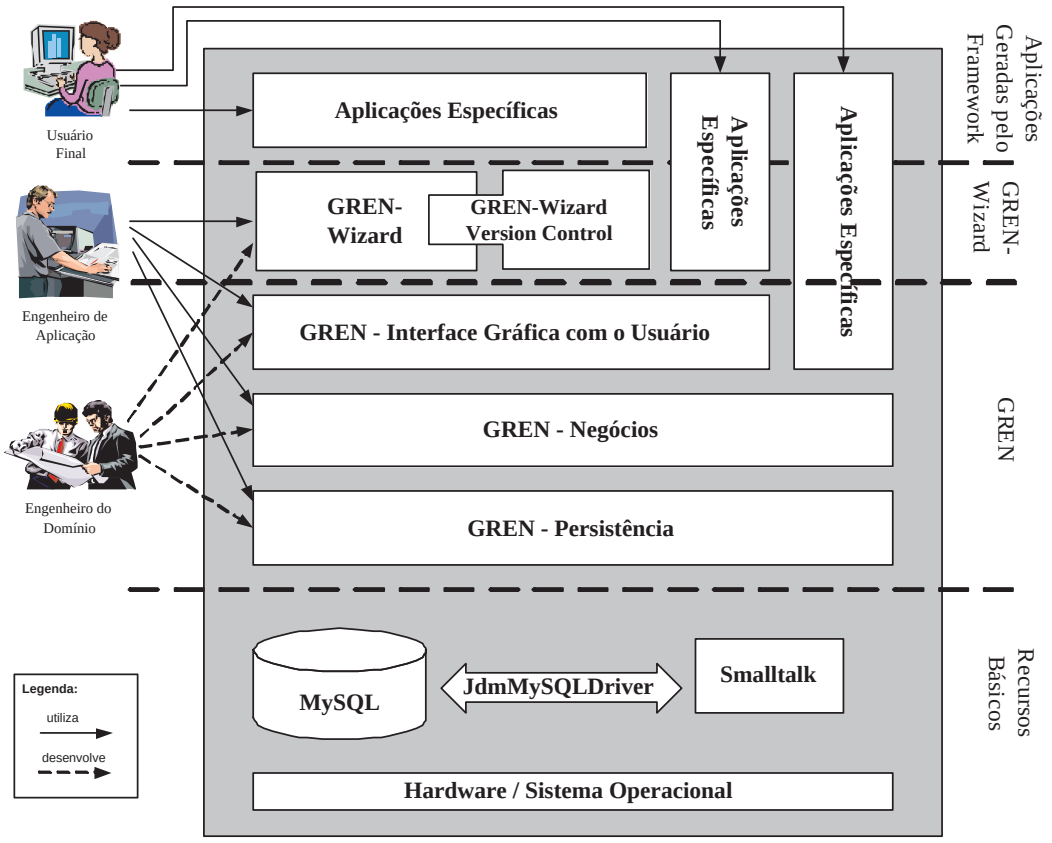


Figura 6.4: Arquitetura do framework GREN (adaptada de Braga (2003))

De acordo com a Figura 6.5, o engenheiro de aplicação deve utilizar inicialmente a ferramenta GREN-Wizard para gerar uma primeira versão da aplicação. Nessa ferramenta, ele especifica a aplicação informando os padrões da GRN utilizados para modelá-la. Posteriormente, a aplicação é criada pelo gerador de código e fica disponível para ser utilizada pelo seu usuário final. Qualquer modificação manual realizada no código fonte da aplicação, visando atender novos requisitos ou regras do negócio não fornecidos pelo framework, deve ser feita por meio da ferramenta *GREN-WizardVersionControl*, que armazena cada modificação em sua base de dados. Quando houver necessidade de inserir novos requisitos na aplicação, fornecidos pelo framework, o engenheiro de aplicação deve utilizar a ferramenta GREN-Wizard novamente. Essa ferramenta gera a nova versão da aplicação, com os novos requisitos herdados, juntamente com todas as modificações realizadas anteriormente no código fonte. Todas as versões da aplicação podem ser utilizadas pelo usuário final.

A ferramenta *GREN-WizardVersionControl* armazena apenas as modificações realizadas em cada versão das aplicações. Isso porque existe um histórico de instanciação do framework, que é considerado pela ferramenta GREN-Wizard, que armazena a funcionalidade de cada aplicação gerada. Com isso, é possível obter automaticamente informação da versão mais antiga da aplicação.

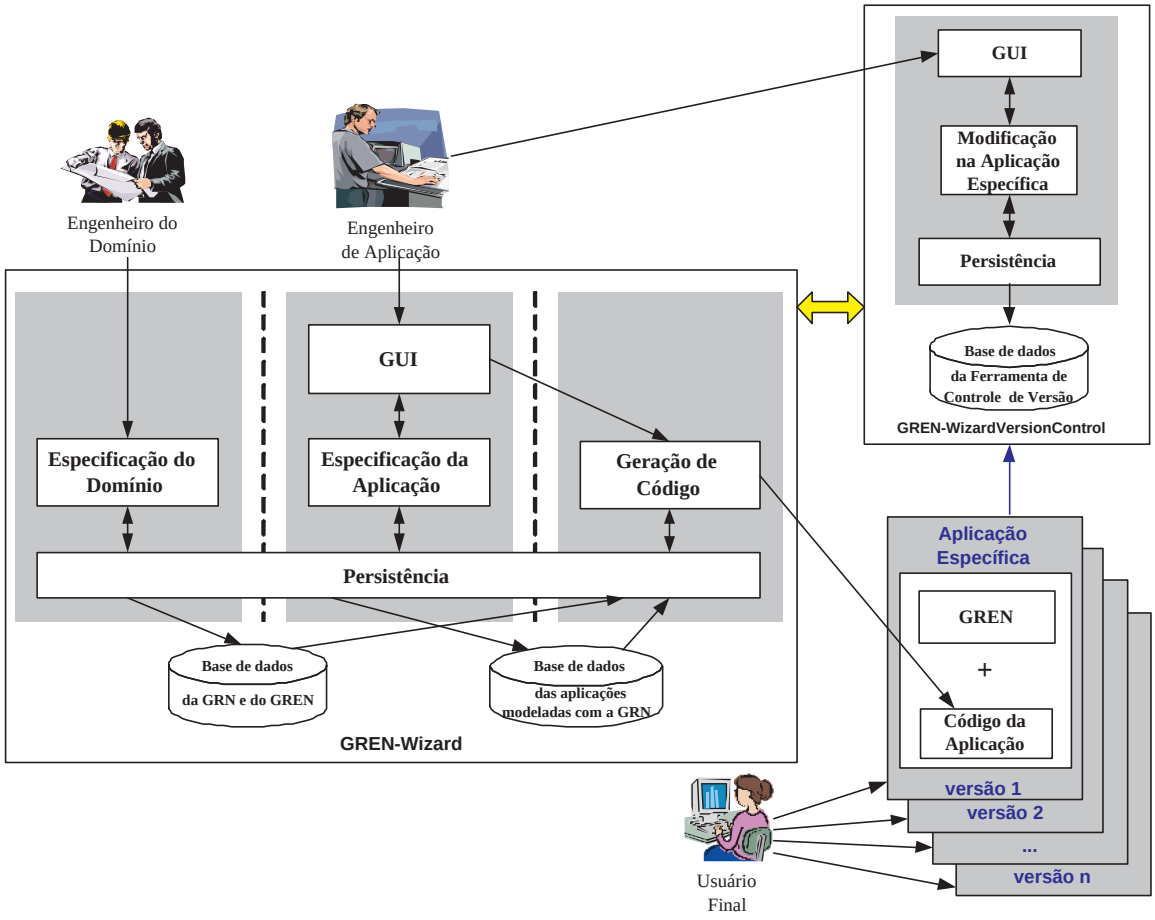


Figura 6.5: Arquitetura das ferramentas GREN-Wizard e *GREN-WizardVersionControl*

6.3 Evolução da Ferramenta GREN-Wizard

Para facilitar a instanciação do framework GREN há um ferramenta de apoio, denominada GREN-Wizard (Braga e Masiero, 2003), como mencionado na Subseção 2.4.2.1 do Capítulo 2, que solicita os padrões da GRN usados na modelagem da aplicação e gera as classes em Smalltalk e as tabelas correspondentes no banco de dados relacional MySQL.

Para desenvolver a ferramenta *GREN-WizardVersionControl* foi necessário também evoluir a ferramenta GREN-Wizard e integrá-la à ferramenta *GREN-WizardVersionControl*, para que as alterações realizadas no código fonte de uma determinada aplicação fossem consideradas também nas próximas instanciações do framework para tal aplicação. Com isso, durante a geração de uma nova versão da aplicação, as classes, métodos e atributos que foram removidos, inseridos ou atualizados em versões anteriores da aplicação, e que estão armazenados na base de dados da ferramenta *GREN-WizardVersionControl*, são detectados. Nesse procedimento, a ferramenta GREN-Wizard considera a última ocorrência, ou seja, a versão mais atual de cada elemento da aplicação armazenada na base de dados da *GREN-WizardVersionControl*.

Algumas classes existentes do framework GREN (`PersistentObject`) e da ferramenta de instanciação GREN-Wizard (`GrenWizardCodeGenerator`, `GrenWizardGUI`, `GrenApplication`, `AttributeMappingForm` e `AttributeListForm`) tiveram que ser modificadas para possibilitar o desenvolvimento da ferramenta *GREN-WizardVersionControl*, e estão apresentadas no diagrama de classes da Figura 6.6 com preenchimento na cor cinza. A classe `GrenWizardDifferatorTool` [2] foi criada para mostrar os trechos de código fonte dos métodos da aplicação, herdados do framework, mas que foram modificados pelo engenheiro de aplicação. Isso acontece durante a instanciação de uma nova versão da aplicação com o apoio da ferramenta GREN-Wizard.

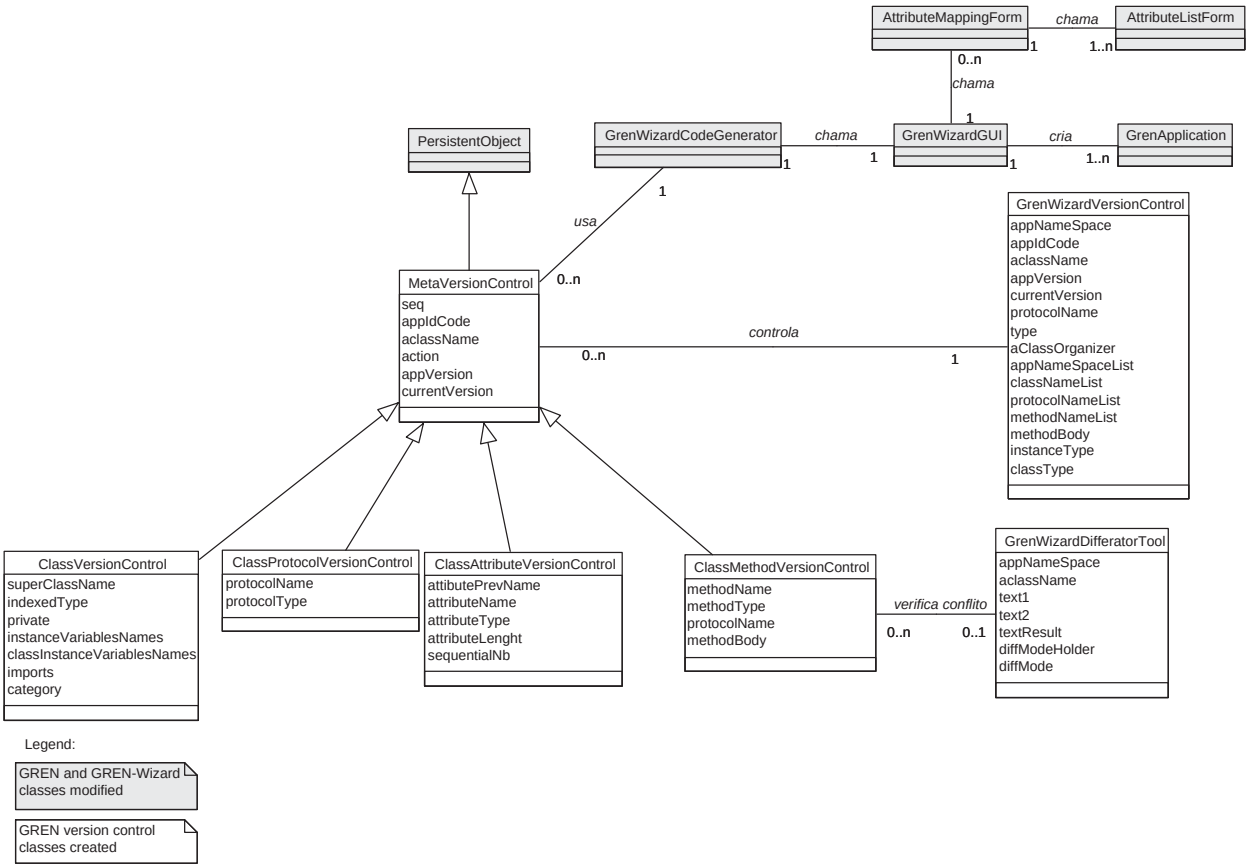


Figura 6.6: Diagrama de classes da evolução da GREN-Wizard

As inserções e remoções de código fonte são resolvidas automaticamente pela ferramenta *GREN-WizardVersionControl* durante a instanciação do framework. No entanto, as modificações nos métodos herdados do framework são identificadas pela ferramenta como conflito entre o código fonte do método do framework e o da aplicação e deve ser resolvido pelo engenheiro da aplicação em tempo de instanciação do framework. Quando a ferramenta *GREN-WizardVersionControl* detecta conflitos, é apresentada uma tela para o engenheiro da aplicação, Figura 6.7 (lado esquerdo), que deve informar o conteúdo do método que a

[2]Baseada na classe `Differator` do VisualWorks que identifica diferenças entre string e entre código fonte escrito em Smalltalk.

ferramenta GREN-Wizard deve considerar na versão da aplicação sendo criada. Para isso, é necessário copiar e colar o conteúdo correto na caixa de texto *Resulting Method*, Figura 6.7 (lado direito). Em seguida, deve-se clicar no botão *Accept* para compilar as linhas de código fonte informadas e dar prosseguimento à instanciação do framework.

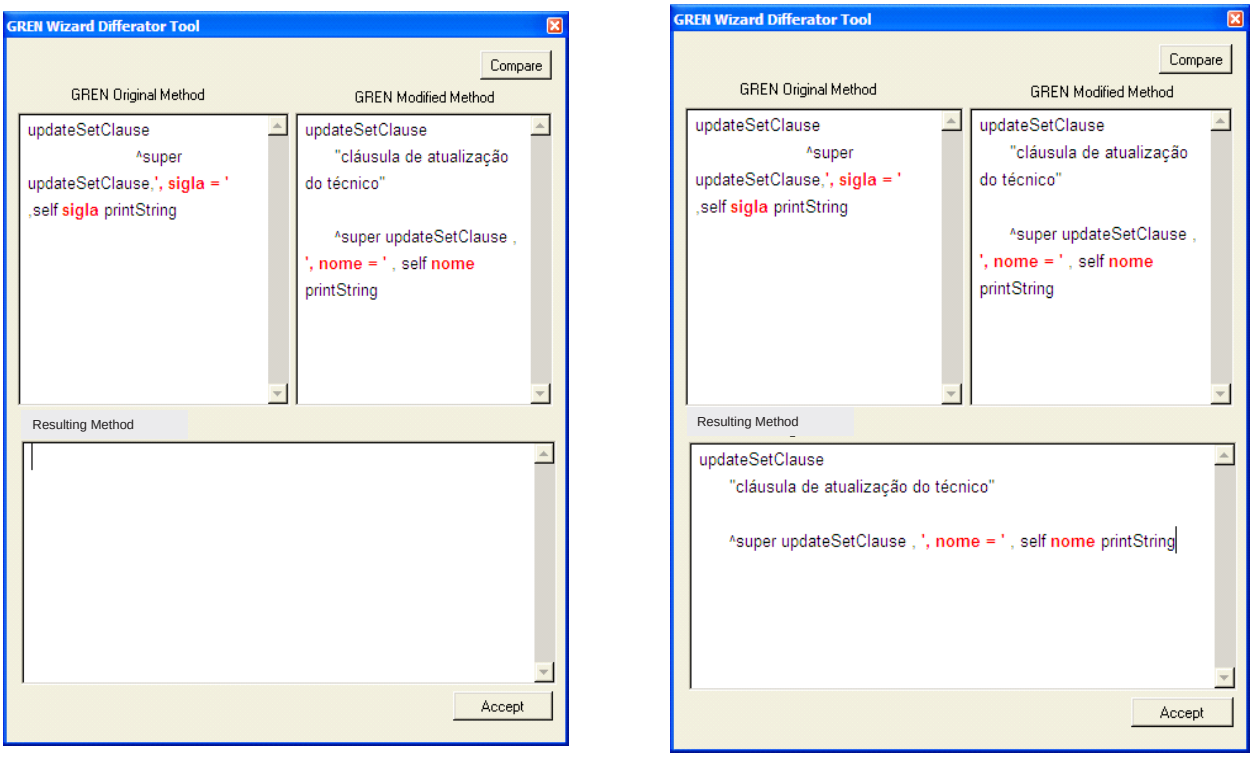


Figura 6.7: Tela para solução de conflito

Como é importante manter os registros das tabelas de versões anteriores da aplicação nas novas versões, optou-se por realizar uma evolução na função da ferramenta GREN-Wizard que cria a base de dados e os *scripts*, em MySQL, das tabelas da aplicação sendo gerada. Com essa evolução, o engenheiro de aplicação tem a opção de gerar uma nova base de dados da aplicação ou manter a atual e apenas atualizar a estrutura das tabelas modificadas, para não perder os dados da aplicação inseridos na versão anterior.

6.4 Um Exemplo de Uso da Ferramenta GREN-WizardVersionControl

Para exemplificar o uso da ferramenta *GREN-WizardVersionControl*, foi criado um sistema de biblioteca com funcionalidade semelhante a do sistema alvo resultante do estudo de caso de reengenharia apresentado no Capítulo 3. Por isso, o diagrama de classes elaborado nesse estudo de caso é utilizado no exemplo de uso discutido nesta seção. Ressalta-se que o termo sistema é aqui utilizado como sinônimo de aplicação.

Inicialmente, gera-se a primeira versão do sistema por meio da ferramenta GREN-Wizard. Em seguida, faz-se alterações manuais nessa versão com o apoio da ferramenta *GREN-WizardVersionControl* para refletir comportamentos do sistema legado não fornecidos pelo framework e, posteriormente, gera-se uma nova versão do sistema com o apoio da GREN-Wizard para herdar um outro requisito funcional fornecido pelo framework, necessário para o sistema. Nessa última versão gerada, as alterações manuais feitas anteriormente são incorporadas automaticamente.

Como mencionado, a primeira versão do sistema foi criada com o apoio da ferramenta GREN-Wizard, em que especificou-se o sistema por meio dos padrões da GRN utilizados durante a sua modelagem. O diagrama de classes do sistema resultante da modelagem feita na reengenharia é ilustrado na Figura 6.8. Para obtê-lo foram aplicados os padrões 1 (IDENTIFICAR O RECURSO), 2 (QUANTIFICAR O RECURSO) e 4 (LOCAR O RECURSO) da GRN. As classes referentes aos padrões da GRN estão ilustradas na parte superior da figura. As classes do sistema são representadas como subclasses das classes dos padrões, com atributos e métodos específicos do sistema, e estão apresentadas na parte inferior da figura. Atributos enumerados com valor a partir de um objeto de uma classe e atributos multivalorados com valores a partir de objetos de uma classe estão representados no diagrama por meio de classes com os estereótipos <<*tipo enumerado*>> (`Curso` e `Editora`) e <<*tipo multivalorado*>> (`Assunto` e `Autor`), respectivamente. Atributos enumerados com valor a partir de valores pré-definidos são representados como atributos com tipo *TipoEnumerado* (nível da classe `Curso`) e com a especificação dos valores entre colchetes. Anotações da UML (retângulos) no diagrama de classes do sistema são usadas para indicar a variante do padrão que foi considerada na modelagem.

Na Figura 6.9 é apresentada a tela da ferramenta GREN-Wizard durante a especificação do sistema de biblioteca, mais especificamente, durante a aplicação do padrão 4 da GRN - LOCAR O RECURSO. No caso específico desse sistema, o *Recurso* é o Livro, o *Destino* é o Aluno e a *Locação do Recurso* é o Empréstimo. A ferramenta GREN-Wizard permite que novos atributos, inerentes ao sistema, sejam especificados nas classes participantes dos padrões. Assim, de acordo com o diagrama de classes elaborado durante a reengenharia, a classe `Livro` possui atributos específicos e portanto foram especificados, conforme apresentado na Figura 6.10. Depois de finalizar a especificação, a primeira versão do sistema alvo e os *scripts* das suas tabelas foram gerados pela ferramenta GREN-Wizard (botões *Gerar aplicação* e *Criar tabelas MySQL*, respectivamente). Na Figura 6.11 são apresentados a tela de empréstimo de livro e o *script* da tabela empréstimo gerados.

Posteriormente, foi necessário realizar algumas modificações no código fonte do sistema para torná-lo com funcionalidade semelhante à do sistema legado. Para isso, a ferramenta *GREN-WizardVersionControl* foi utilizada. O *layout* da tela de empréstimo de livro teve que ser alterado para se adequar à interface do legado, pois no legado não havia os campos de totalizações localizados na parte inferior da tela de empréstimo de livro (Figura 6.11).

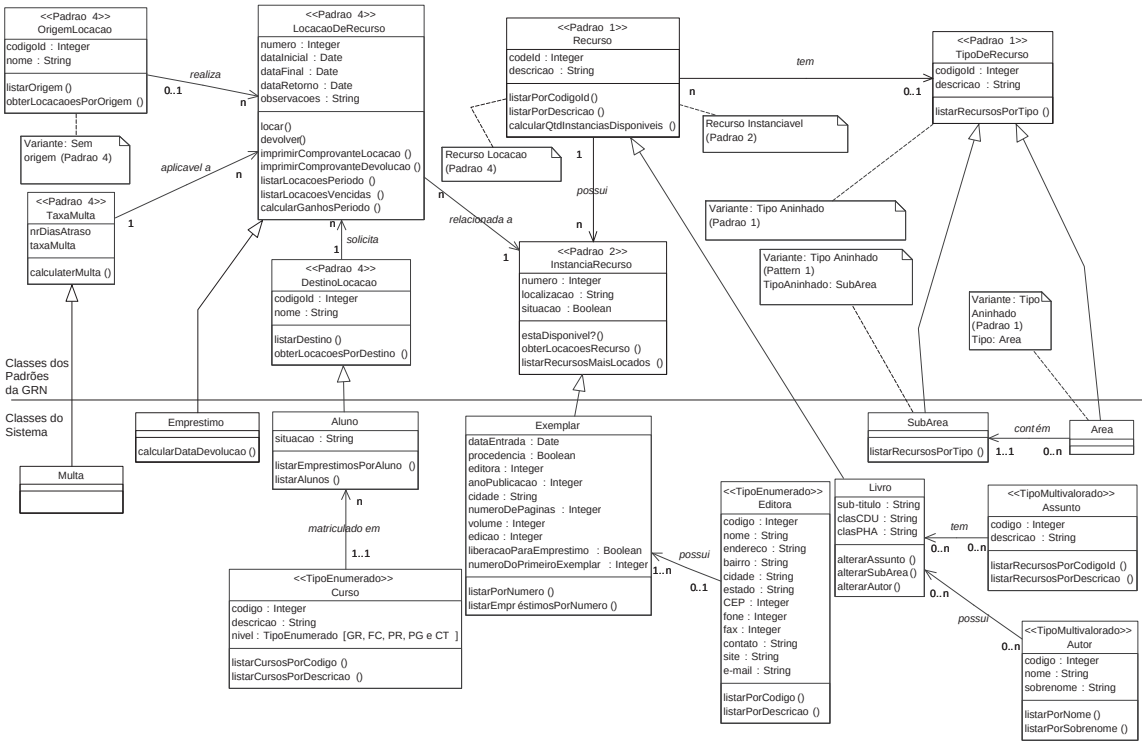


Figura 6.8: Diagrama de classes do sistema de biblioteca (Chan et al., 2003)

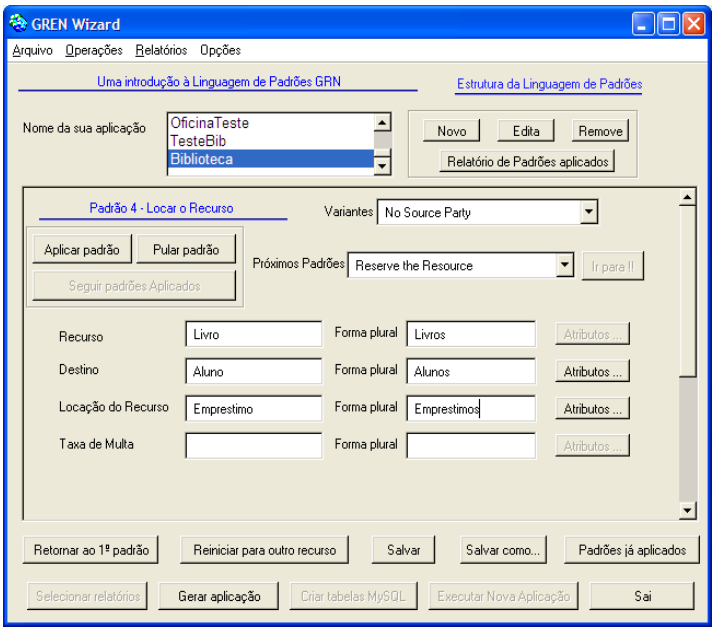


Figura 6.9: Tela da ferramenta GREN-Wizard durante a especificação do sistema de biblioteca

Na primeira versão do sistema alvo gerada pela ferramenta GREN-Wizard, o método que especifica essa tela não foi sobreposto pois nenhum atributo novo foi inserido na classe `Emprestimo`. No entanto, para realizar a alteração desejada, o método de especificação da tela (`windowSpec`) teve que ser sobreposto na classe `EmprestimoForm` do sistema e teve seu nome modificado para `EmprestimoSpec`, conforme apresentado na Figura 6.12. Nessa figura

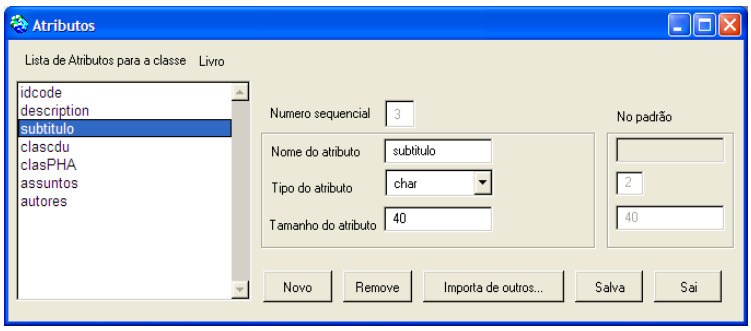


Figura 6.10: Inserção de novos atributos na classe `Livro`

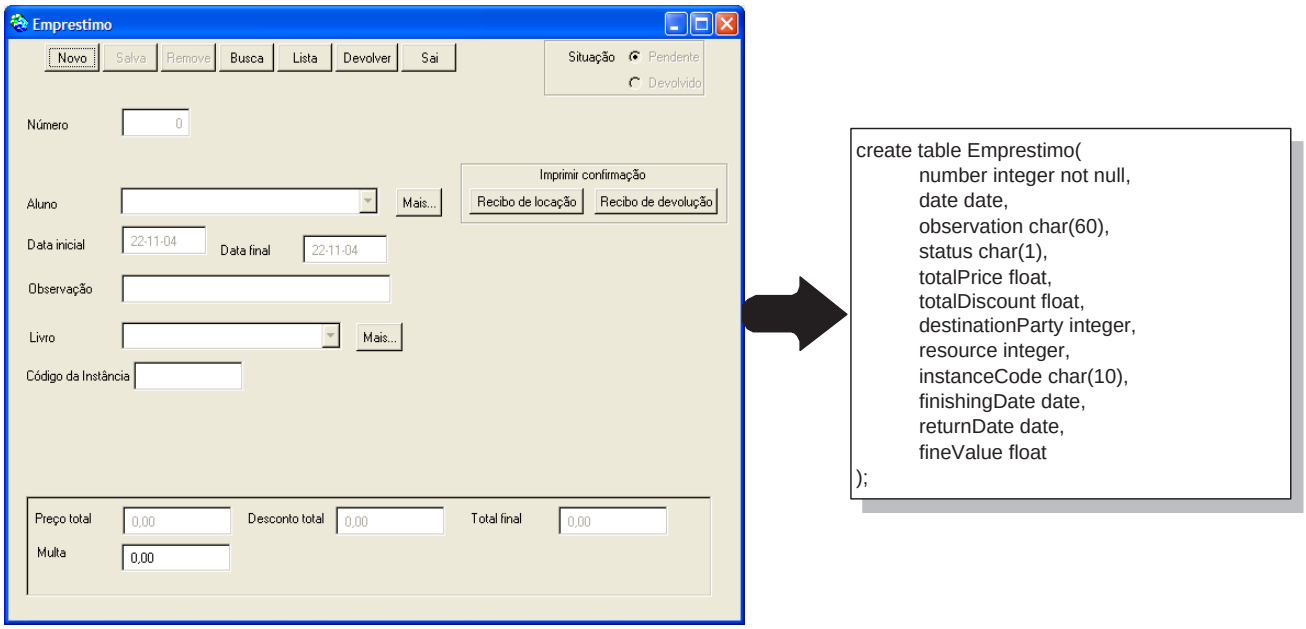


Figura 6.11: Tela de empréstimo de livros e *script* da tabela `Emprestimo`

também é apresentado o armazenamento da modificação no código fonte do sistema alvo na base de dados da ferramenta *GREN-WizardVersionControl*.

Como a modificação foi inserir um método (`EmprestimoSpec`) em uma determinada classe (`EmprestimoForm`), o armazenamento foi feito na tabela `ClassMethodVersionControl` da base de dados da ferramenta, contendo o código do sistema especificado na ferramenta GREN-Wizard (`appIdCode`); o nome da classe em que o método foi inserido (`className`); o nome do método (`methodName`); o tipo do método (`appIdCode`), ou seja, de instância (**i**) ou de classe (**c**); o nome do protocolo de métodos em que o método foi inserido (`protocolName`); o código fonte do método (`methodBody`); a ação realizada (`action`), ou seja, inserção/alteração (**i**) ou remoção (**r**); o número da versão do sistema gerada pela ferramenta GREN-Wizard (`appVersion`) e o número da nova versão do sistema que está sendo modificada (`version`). Na Figura 6.13 é apresentado o novo *layout* da tela de empréstimo sem os campos de totalizações.

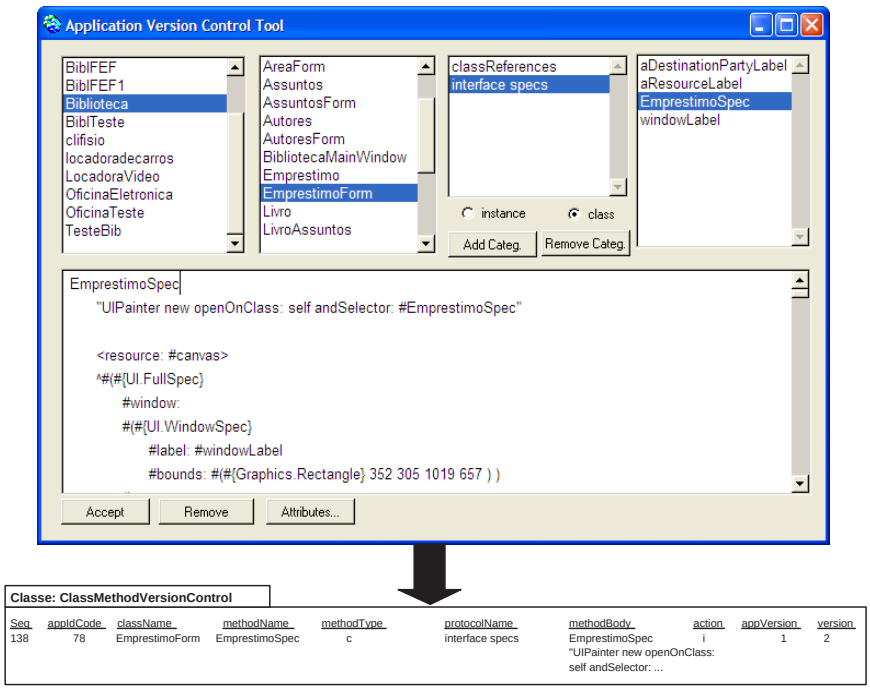


Figura 6.12: Sobreposição do método `EmprestimoSpec` na classe `EmprestimoForm`

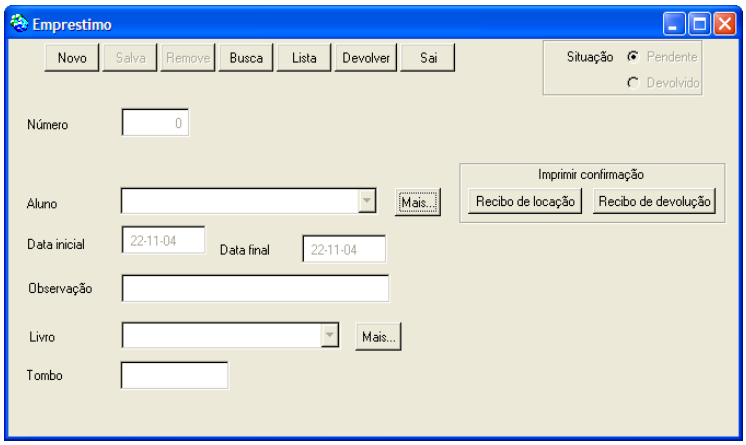


Figura 6.13: Novo *layout* da tela de empréstimo de livros

Como conseqüência da alteração no nome do método de especificação da tela de empréstimo, o método `emprestimo` da classe `BibliotecaMainWindow` também foi alterado. Essa alteração foi também armazenada na base de dados da ferramenta *GREN-WizardVersionControl*, conforme apresentado na Figura 6.14. Esse método está associado a opção do menu que abre a tela de empréstimo.

Como comentado na Seção 6.2 e como apresentado na Figura 6.15, a ferramenta *GREN-WizardVersionControl* não permite que elementos herdados do framework GREN, ou seja, pertencentes aos padrões aplicados durante a especificação do sistema gerado pela ferramenta GREN-Wizard, sejam removidos. Isso garante que o sistema continue funcionando adequadamente, mesmo depois de realizar modificações nele.

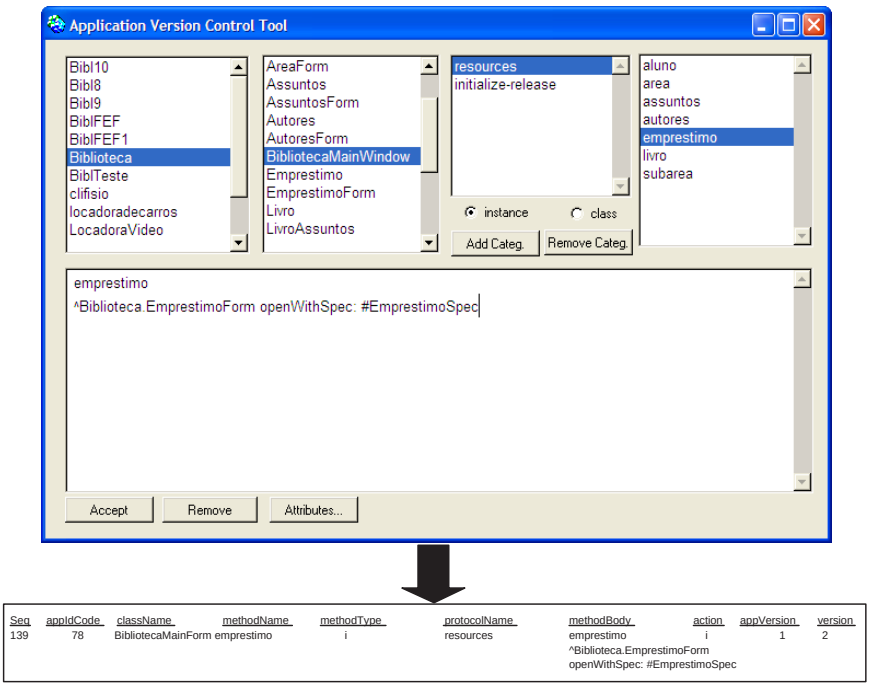


Figura 6.14: Alteração do método `emprestimo` da classe `BibliotecaMainWindow`

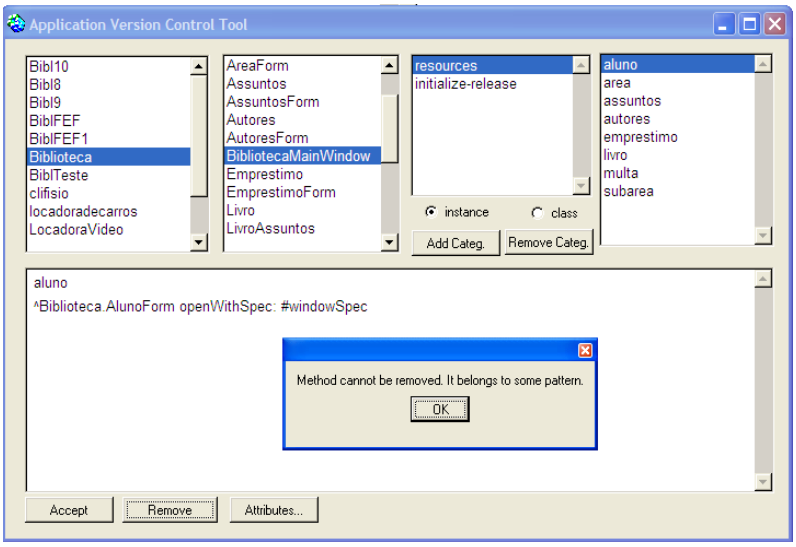


Figura 6.15: Consiste remoção de elementos herdados do framework

Durante a reengenharia, o usuário do sistema legado solicitou a incorporação de uma regra do negócio relacionada à cobrança de multa para aqueles leitores que devolvem o livro com atraso. Como a implementação dessa regra é fornecida pelo framework, optou-se por instanciá-lo novamente. A regra do negócio foi especificada na ferramenta GREN-Wizard, adicionando-se o nome do papel "Taxa de Multa" na aplicação do padrão 4 da GRN (Figura 6.16). Após terminar a especificação da multa, o framework foi instanciado novamente (botão *Gerar aplicação*).

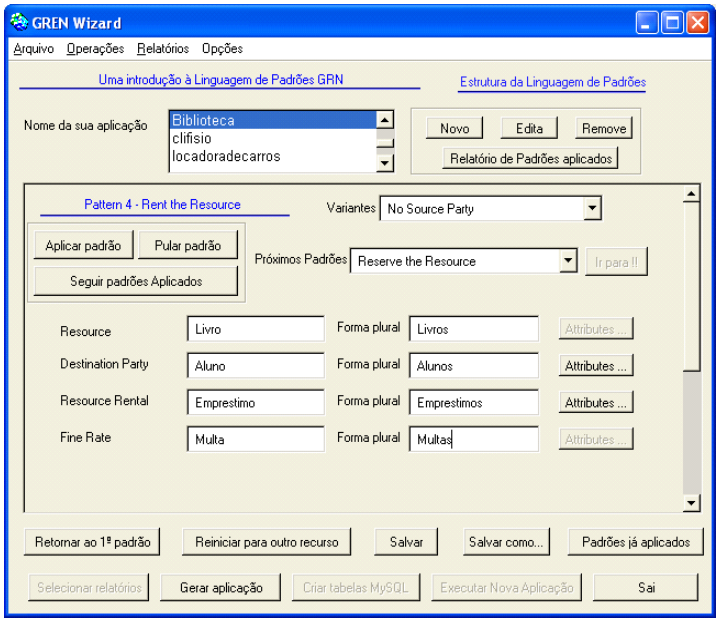
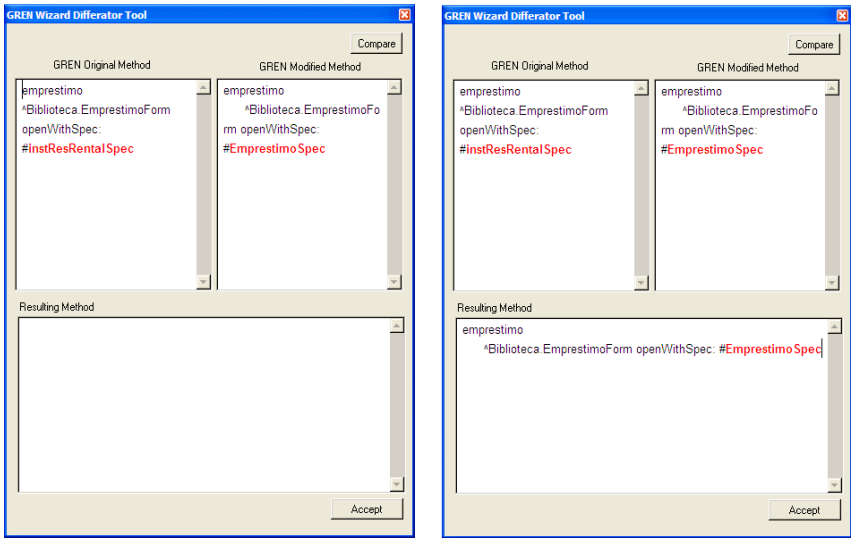


Figura 6.16: Tela da ferramenta GREN-Wizard durante a especificação da multa

Durante a criação da nova versão do sistema, a ferramenta *GREN-WizardVersionControl* detectou, em tempo de instanciação do framework GREN, a modificação realizada no método `emprestimo` da classe `BibliotecaMainWindow` como conflito entre o conteúdo do método herdado do framework e o do sistema e apresentou ao engenheiro de aplicação a tela *GREN-Wizard Differator Tool*, ilustrada na Figura 6.17(a). O engenheiro de aplicação teve que solucionar esse conflito, selecionando o conteúdo desejado do método (Figura 6.17(b), parte inferior) e clicando no botão *Accept* para prosseguir a instanciação. Como as inserções e remoções de código fonte são resolvidas automaticamente pela ferramenta *GREN-WizardVersionControl*, o método `EmprestimoSpec` foi inserido automaticamente na nova versão do sistema.

Antes de gerar o *script* das tabelas da nova versão do sistema de biblioteca, a ferramenta *GREN-WizardVersionControl* apresentou uma caixa de diálogo para que o engenheiro de aplicação decidisse por gerar uma nova base de dados do sistema ou manter a atual e apenas atualizar a estrutura modificada para não perder os dados do sistema inseridos na sua versão anterior (Figura 6.18). Decidiu-se manter a base de dados anterior e apenas atualizar a estrutura das tabelas que foram modificadas.

Na Figura 6.19 é apresentada a tela de cadastro de multa e o *script* da respectiva tabela, ambos criados pela ferramenta GREN-Wizard na segunda instanciação do framework.



(a) Tela *GREN-Wizard Differator Tool* (b) Tela de solução de conflito

Figura 6.17: Tela de conflito da *GREN-WizardVersionControl*

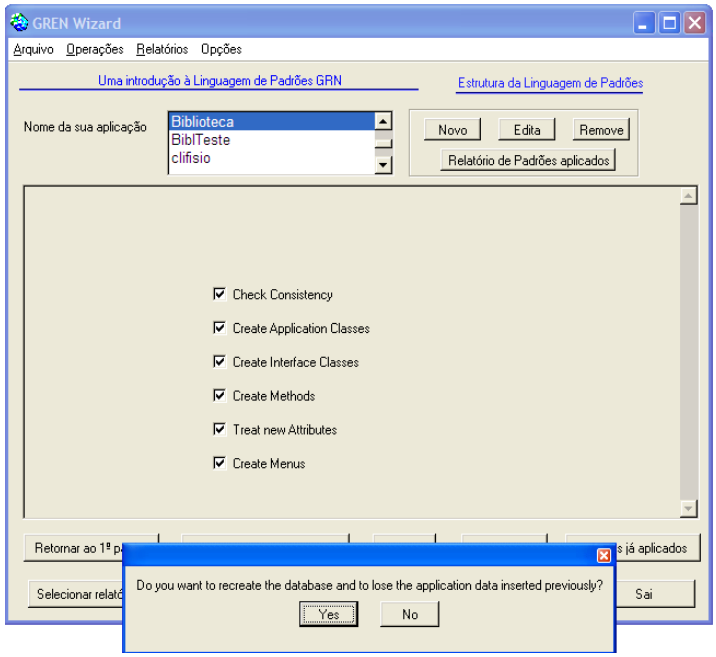


Figura 6.18: Mensagem da ferramenta *GREN-WizardVersionControl* antes de gerar a base de dados do sistema

6.5 Considerações Finais

GREN-WizardVersionControl é útil não apenas na aplicação do ARA como também no desenvolvimento de novos sistemas com o apoio do framework GREN, seguindo a abordagem iterativa e incremental praticada pelos métodos ágeis. Por meio dessa ferramenta, o engenheiro de aplicação armazena um histórico de todas as mudanças feitas em cada versão

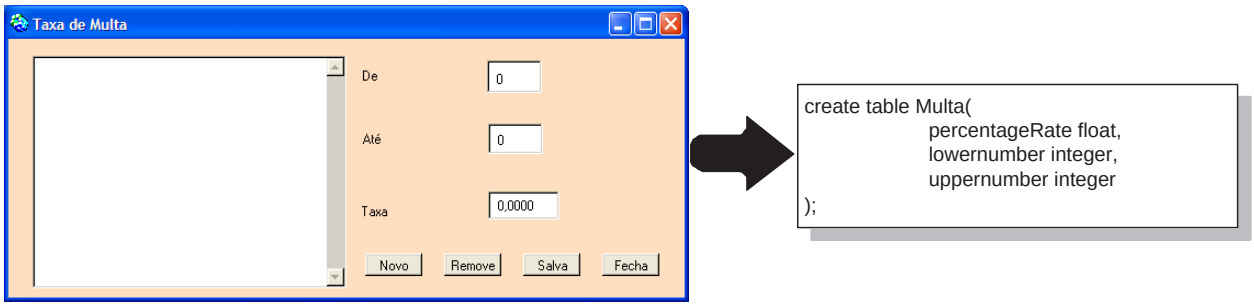


Figura 6.19: Tela de cadastro de multa e o *script* da respectiva tabela

do sistema. Essas mudanças são incorporadas às novas versões de cada sistema, gerado pela ferramenta GREN-Wizard, com o mínimo de intervenção do engenheiro de aplicação.

Com o uso da ferramenta *GREN-WizardVersionControl* para criar o sistema de biblioteca, apresentado na Seção 6.4, pôde-se notar o apoio à reengenharia iterativa e incremental, baseada no framework GREN, ou seja, o framework pode ser instanciado quantas vezes for necessário e as linhas de código fonte inseridas nas versões do sistema, resultantes das instanciações do framework, não são perdidas. Essa ferramenta é disponibilizada juntamente com o arcabouço definido.

No próximo capítulo apresentam-se as conclusões obtidas com o trabalho desenvolvido nesta tese, destacando as contribuições e limitações, bem como sugestões de trabalhos futuros decorrentes desta pesquisa.

Capítulo

7

Conclusão

7.1 Considerações Iniciais

Neste capítulo apresenta-se um resumo do trabalho realizado, destacam-se as contribuições obtidas para a área de Engenharia de Software e listam-se as suas limitações. Além disso, citam-se também sugestões de trabalhos futuros, decorrentes da pesquisa realizada nesta tese.

7.2 Resumo do Trabalho Realizado

Neste trabalho foram criados diversos produtos que colaboraram para alcançar o objetivo desta tese citado na Seção 1.3 do Capítulo 1.

Definiu-se um arcabouço de reengenharia ágil baseado em framework, denominado ARA, para apoiar a reengenharia de sistemas legados procedimentais para o paradigma orientado a objetos. Para permitir a efetividade desse arcabouço diversos recursos foram criados e foram a ele associados. Um processo ágil de reengenharia baseado em framework, denominado PARFAIT (Capítulo 3 desta tese), foi criado para viabilizar a aplicabilidade do arcabouço. A partir dos resultados obtidos em três estudos de caso apresentados no Capítulo 3, utilizando esse processo e, conseqüentemente, o arcabouço ARA, com o apoio do framework GREN (Subseção 2.4.2.1 do Capítulo 2), observou-se a necessidade de definir outros recursos e associá-los a tal arcabouço: o processo de evolução de frameworks de

aplicação PREF (Capítulo 4), a abordagem de reúso de teste ARTe (Capítulo 5) e a ferramenta *GREN-WizardVersionControl* (Capítulo 6).

O PREF surgiu da necessidade de evoluir frameworks de aplicação, que não cobrem requisitos funcionais ou não funcionais inerentes ao seu domínio e presentes nos sistemas legados submetidos à reengenharia com o apoio do PARFAIT. Ressalta-se que a necessidade de uso desse processo também pode ser observada durante o desenvolvimento de software baseado em frameworks de aplicação. A abordagem ARTe surgiu da necessidade de reduzir o tempo gasto com atividades de VV&T na reengenharia de software. Para isso, recursos de teste são agregados aos padrões da linguagem de padrões, usada na construção do framework de aplicação, que é utilizado como apoio computacional pelo PARFAIT. A ferramenta *GREN-WizardVersionControl* surgiu da necessidade de fornecer subsídios à característica iterativa do arcabouço ARA, com o apoio do framework GREN.

Além dos produtos mencionados, um pacote de experimentação foi esboçado (Apêndice A) para analisar processos de reengenharia baseados em frameworks, no caso, o PARFAIT; enfocando principalmente as fases de definição e planejamento.

Estudos de caso e exemplos de uso foram feitos com cada um dos produtos definidos nesta tese e resultados apontaram a viabilidade de uso desses produtos. Estudos controlados devem ser conduzidos para a obtenção de resultados com significância estatística.

7.3 Contribuições

A principal contribuição deste trabalho é a definição de um arcabouço de reengenharia ágil baseado em framework cuja construção tenha sido baseada em linguagem de padrões, que proporciona reúso em diversos níveis de abstração, como ilustrado na Figura 7.1. Para permitir o uso desse arcabouço definiu-se um processo ágil de reengenharia também baseado em framework (Seção 3.3 do Capítulo 3), que seguiu as principais idéias de tal arcabouço.

O entendimento e a elaboração da documentação orientada a objetos do sistema legado podem ser reutilizados a partir da linguagem de padrões de análise, usada na construção do framework de aplicação. Isso porque a linguagem de padrões fornece conhecimento do domínio ao qual o sistema legado pertence e também soluções de análise (por exemplo, na forma de diagrama de classes) de problemas recorrentes que são encontrados em tal domínio. Os recursos de teste agregados aos padrões da linguagem de padrões de análise com o apoio da abordagem ARTe, definida nesta tese, colaboram para o reúso no nível de teste, desde o início da reengenharia, possibilitando a "reengenharia guiada por teste". A implementação e o projeto do sistema alvo são reutilizados a partir do framework utilizado. As adaptações feitas em cada versão "intermediária" do sistema alvo são incorporadas em sua subseqüente versão com o apoio da ferramenta de controle de versão *GREN-WizardVersionControl*, colaborando

para o reúso no nível de manutenção. Assim, o reúso na reengenharia em diversos níveis pode contribuir com a redução de custos e esforços associados.

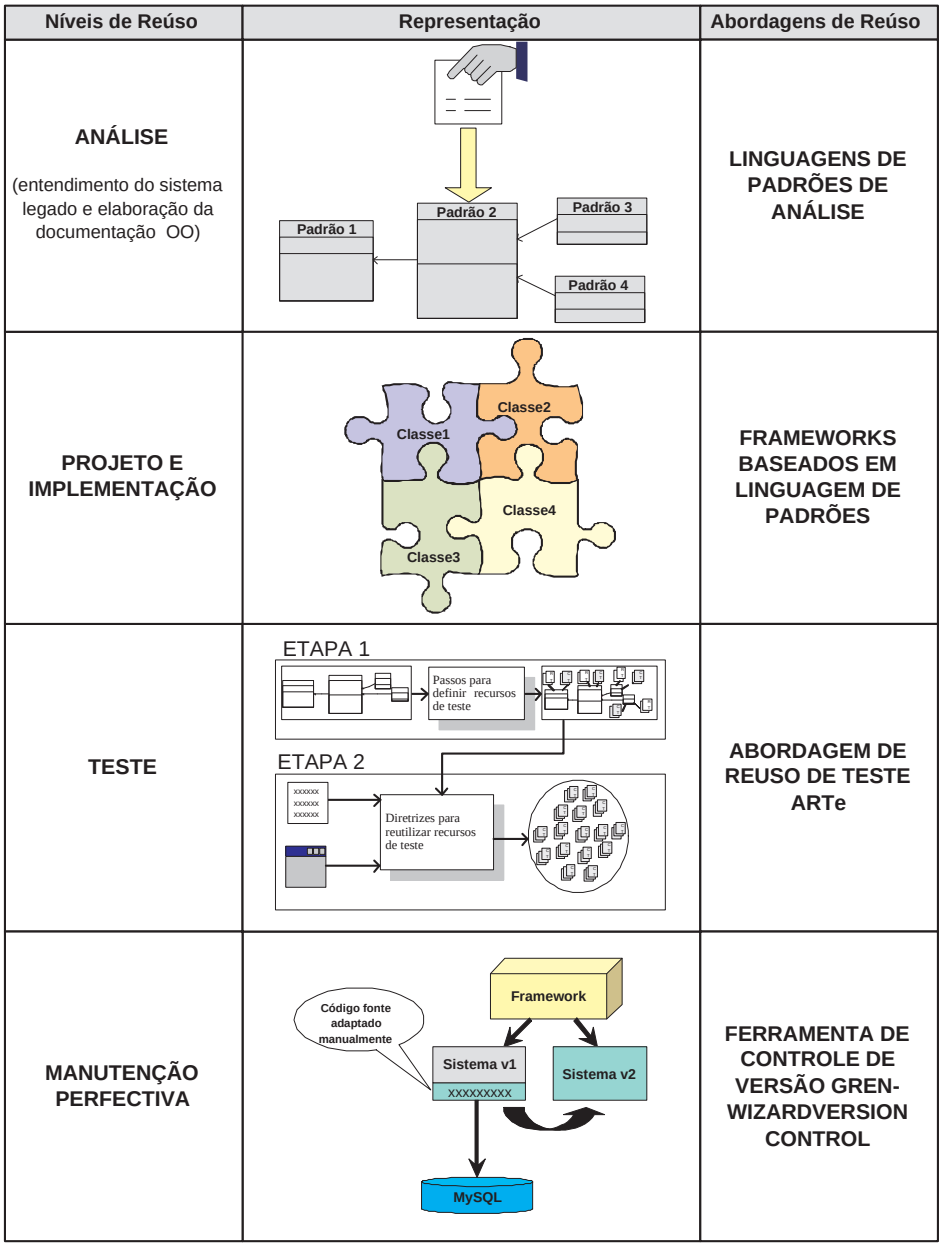


Figura 7.1: Formas de reúso do PARFAIT

As práticas de métodos ágeis embutidas nas atividades do PARFAIT colaboram para que o sistema alvo possa ser entregue dentro do prazo e custos previamente estimados. Além disso, notou-se que esse processo pretende minimizar alguns dos riscos associados à reengenharia, discutidos por Rosenberg (1996), como justificado na Seção 3.3 do Capítulo 3.

Dessa forma, PARFAIT e, conseqüentemente, o arcabouço ARA, preenche diversas carências levantadas na Seção 1.1 do Capítulo 1, ou seja: 1) de processos e abordagens de reengenharia com apoio computacional efetivo na reengenharia e com atividades VV&T associadas, 2) de abordagens de teste que associam recursos de teste a padrões, 3) do uso

de padrões de análise para apoiar o entendimento do sistema legado, 4) de processos de reengenharia que utilizam frameworks de aplicação e 5) de processos de reengenharia que utilizam práticas ágeis.

Salienta-se que o arcabouço ARA pode ser adaptado para apoiar também o desenvolvimento de software. Resumidamente, o documento de requisitos é elaborado a partir de entrevistas com os usuários do sistema. O diagrama de classes é criado com o apoio dos padrões da linguagem de padrões de análise, a partir dos requisitos definidos. Os casos de teste do sistema são reutilizados daqueles agregados aos padrões da linguagem de padrões, colaborando para o desenvolvimento guiado por teste (*test-driven development* (Beck, 2002)). Como na reengenharia, o projeto e a implementação do novo sistema são obtidos a partir da instanciação do framework. Requisitos e regras do negócio, específicos do sistema, são implementados manualmente no código fonte do sistema. Ressalta-se que todo o desenvolvimento pode ser feito de maneira iterativa e incremental, sendo que os clientes e/ou usuários do sistema priorizam os requisitos que devem ser considerados em cada iteração do processo e também participam de todo o desenvolvimento do software, validando os artefatos produzidos. Em geral, os requisitos que têm mais valor de negócio são implementados nas primeiras versões do novo sistema, a fim de diminuir os riscos associados no desenvolvimento. Diferentemente do PARFAIT no contexto de reengenharia, a implantação do novo sistema pode ser feita gradativamente, até atingir todo o sistema. A migração dos dados de versões anteriores do novo sistema para uma nova versão é apoiada pela ferramenta *GREN-WizardVersionControl*.

O desenvolvimento da ferramenta *GREN-WizardVersionControl* permitiu que o framework GREN atendesse às características iterativa e incremental do ARA. Essa ferramenta pode ser também utilizada no desenvolvimento de sistemas com o apoio de processos que possuem tais características. Apesar da ferramenta *GREN-WizardVersionControl* ser específica a um único framework, a sua estrutura geral pode ser utilizada como base pela comunidade ágil quando o interesse é utilizar frameworks para apoiar o desenvolvimento ágil.

Outra contribuição desta tese é a definição do processo de evolução de frameworks de aplicação PREF. Apesar do processo PREF ter alguns passos que são similares a qualquer outro processo de manutenção de software convencional, fornece contribuição relacionada ao controle de variabilidade de framework. Isso preenche outra carência, identificada na Subseção 2.4.1 do Capítulo 2, ou seja, ausência de processos de evolução específicos para frameworks de aplicação. O processo é utilizado por engenheiros de domínio, quando decide-se evoluir o framework; e por engenheiros de aplicação, quando decide-se evoluir apenas os sistemas gerados a partir da instanciação do framework. O controle de variabilidade é apoiado por um Histórico de Requisitos, que contém requisitos funcionais e não funcionais "não atendidos", "sendo atendidos" ou "atendidos" pelo framework. Esse Histórico de Requisitos é utilizado pelo engenheiro de domínio para apoiar a decisão de evoluir ou não o

framework e também indica a versão do framework em que um determinado requisito, não considerado durante o desenvolvimento do framework, pode ser encontrado.

O pacote de experimentação, apesar de ter sido apresentado parcialmente no Apêndice A, também é outra contribuição da tese, pois contém toda a instrumentação necessária (ou seja, documentos e material de apoio ao treinamento, e formulários para coleta de dados) para apoiar os interessados que desejam conduzir tal experimento, tanto no meio industrial quanto no meio acadêmico. Isso preenche outra carência levantada na Seção 1.1 do Capítulo 1, que está relacionada à ausência de pacotes de experimentação disponíveis no contexto de reengenharia de software. Além disso, o pacote definido contém diversas lições aprendidas durante a operação parcial de um estudo de caso conduzido para validar tal pacote. Esse estudo de caso contribuiu para o refinamento de alguns itens da definição e do planejamento do pacote.

7.4 Limitações do Trabalho Realizado

Algumas limitações dos produtos obtidos nesta tese são provenientes, principalmente, do arcabouço ARA ter sido utilizado apenas com o apoio do framework GREN. Conseqüentemente, o PARFAIT foi utilizado em alguns estudos de caso de reengenharia apenas com o apoio computacional do GREN e o PREF foi também aplicado somente durante algumas evoluções desse framework. O mesmo ocorreu com a abordagem ARTe, que foi aplicada somente na linguagem de padrões GRN, utilizada na construção do GREN.

PARFAIT foi utilizado apenas com sistemas de pequeno porte, portanto a sua escalabilidade para sistemas de maior porte não está consolidada. Estudos de caso específicos para observar esses aspectos devem ser conduzidos.

PREF não pode ser considerado genérico, pois a experiência com a evolução de um framework pertencente ao domínio de Gestão de Recursos de Negócios não é suficiente para validar e garantir a generalidade de tal processo. Outra limitação desse processo é a ausência de gerenciamento de controle de configuração mais sistemático das versões do framework e das ferramentas de apoio (caso existam), com as respectivas versões dos sistemas gerados.

Com relação à abordagem ARTe, como foi aplicada somente na GRN, não é possível afirmar o quanto ela é flexível. Para isso, deve ser aplicada a outras linguagens de padrões com outras particularidades. Além disso, o uso dessa abordagem por outras pessoas em outros contextos colaborará para o seu refinamento e evolução.

A ferramenta *GREN-WizardVersionControl*, além de ser específica a um único framework, possui outras limitações, citadas na Seção 6.2 do Capítulo 6, dificultando a execução das práticas ágeis "propriedade coletiva do código" e "integração contínua", principalmente porque não fornece controle de sincronização.

Outra limitação está relacionada ao pacote de experimentação. Isso porque, com a condução parcial de um estudo de caso, apresentado na Seção A.0.3 do Apêndice A, não foi possível completar e avaliar completamente o pacote de experimentação.

Estudos de caso e exemplos de uso foram realizados durante o desenvolvimento deste trabalho. No entanto, não possuem significância estatística que comprovem os resultados obtidos. É importante esclarecer que os estudos conduzidos são limitados às pessoas envolvidas, ao ambiente onde foram conduzidos e às condições impostas (por exemplo, tamanho do sistema legado, restrição de tempo dos participantes dos estudos, uso de apenas um framework, uso de apenas uma linguagem de padrões de análise).

7.5 Sugestões de Trabalhos Futuros

Diversas perspectivas futuras de pesquisa foram identificadas para dar continuidade ao trabalho realizado nesta tese:

- Investigação dos processos PARFAIT e PREF e da abordagem ARTe no contexto de outros frameworks de aplicação baseados em linguagens de padrões;
- Investigação de técnicas específicas da Engenharia de Requisitos para apoiar a análise dos requisitos não funcionais do framework e do sistema legado durante a aplicação da atividade "Confrontar as características não funcionais do framework x sistema legado" do PARFAIT.
- Investigação de outras abordagens de reúso existentes, como por exemplo, componentes (Chessman e Daniels, 2001; Gimenes e Huzita, 2005) e padrões de software organizacionais (Coplien, 1997; Coplien e Harrison, 2004; Giorgini et al., 2002), para associá-los ao arcabouço ARA.
- Definição de diretrizes para apoiar o gerenciamento de configuração de frameworks e dos sistema gerados por eles.
- Evolução da ferramenta *GREN-WizardVersionControl*.
- Investigação da escalabilidade da ferramenta *GREN-WizardVersionControl* no desenvolvimento e na reengenharia de sistemas de médio e grande porte.
- Definição de estratégias para criar requisitos e casos de teste, baseados em critérios estruturais e em outras técnicas específicas para testar frameworks (Dallal e Sorenson, 2002; Jeon et al., 2002; Tsai et al., 1999), a fim de associá-los às classes de frameworks baseados em linguagens de padrões, como é o caso do GREN.

- Documentação dos recursos de teste da abordagem ARTe em um formato apropriado, por exemplo, XML (*eXtensible Markup Language*) [1] e TCDL (*Test Case Description Language*) [2], para facilitar a recuperação automática por ferramentas e frameworks de teste existentes, por exemplo, XUnit [3].

- Desenvolvimento de uma ferramenta para gerenciar os *hot spots* dos frameworks de aplicação e para apoiar a atividade de controle de variabilidade de frameworks do processo PREF.

- Desenvolvimento de uma ferramenta de apoio à aplicação do processo PARFAIT, para monitorar a execução de cada uma de suas atividades.

- Condução de estudos de caso empíricos para:

  - validar o pacote de experimentação proposto, complementando as fases de análise e interpretação dos dados e de apresentação e empacotamento;
  - observar a escalabilidade do PARFAIT em sistemas legados de médio e grande porte.
  - avaliar o desempenho de cada uma das práticas de métodos ágeis do PARFAIT;
  - observar a aplicabilidade do ARA no desenvolvimento de software e apenas para auxiliar a engenharia reversa de software. Nesse último caso, um processo de engenharia reversa já foi abstraído (Cagnin et al., 2003b), no entanto deve ser refinado por meio de estudos de caso.

[1] `http://www.w3.org/TR/REC-xml/`
[2] `http://www.w3.org/QA/WG/2003/10/tcdl-20031012.html`
[3] `http://xunit.sourceforge.net/`

Referências Bibliográficas

Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. Agile software development methods. review and analysis. ESPOO (Technical Research Centre of Finland)' 2002. VTT Publications n. 478, `http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf`. Acesso em Dezembro/2003, 2002.

Abrahamsson, P.; Warsta, J.; Siponen, M. T.; Ronkainen, J. New Directions on Agile Methods: a Comparative Analysis. In: *ICSE'2003, 25th International Conference on Software Engineering*, p. 244–254, 2003.

Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl, I.; Angel, S. *A Pattern Language*. New York: Oxford University Press, 1977.

Ambler, S. W. *Process patterns: Building large-scale systems using object technology*. Cambridge University Press, 1998.

Ambler, S. W. Validating agile models. *Cutter IT Journal*, v. 15, n. 8, p. 32–39, 2002.

Ambler, S. W.; Jeffries, R. *Agile modeling: Effective practices for extreme programming and the unified process*. first ed. John Wiley & Sons, 2002.

Appleton, B. Patterns and software: Essential concepts and terminology. site, `http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html`. Acesso em Dezembro/2003, 1997.

Arsanjani, A. Component-based development and integration pattern language. In: *EuroPLoP'2001, 6th European Conference on Pattern Languages of Programs*, Irsee, Germany, `http://www.arsanjani.org/pub/CBDLanguage5-10-2001.pdf`. Acesso em Fevereiro/2005, 2001a.

Arsanjani, A. Rule object 2001: A pattern language for adaptive and scalable business rule construction. In: *PLOP'2001, 8th Conference on Pattern Languages of Programs*,

Monticello, Illinois, USA, `http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/AArsanjani1/PLoP2001_AArsanjani1_0.pdf`. Acesso em Fevereiro/2002, 2001b.

Arthur, L. J. *Software evolution: The software maintenance challenge*. Wiley, 1988.

Basili, V.; Lanubile, F.; Shull, F. Investigating maintenance processes in a framework-based environment. In: *ICSM'1998, 14th IEEE International Conference on Software Maintenance*, Bethesda, USA, p. 256–264, 1998.

Baxter, I. D.; Mehlich, M. Reverse engineering is reverse forward engineering. In: *WCRE'1997, 4th Working Conference on Reverse Engineering*, Amsterdam, The Netherlands, p. 104–113, 1997.

Beck, K. *Smalltalk best practice patterns*. Prentice Hall, 1997.

Beck, K. Embracing change with extreme programming. *IEEE Computer*, v. 32, n. 10, p. 70–77, 1999.

Beck, K. *Extreme programming explained: Embrace change*. second ed. Addison-Wesley, 2000.

Beck, K. *Test-driven development: by example*. The Addison-Wesley Signature Series, first ed. Addison-Wesley, 2002.

Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R. C.; Mellor, S.; Schwaber, K.; Sutherland, J.; Thomas, D. Manifesto for agile software development. site, `http://www.agilemanifesto.org`. Acesso em Junho/2003, 2001.

Beck, K.; Johnson, R. Patterns generate architectures. In: *ECOOP'1994, 8th European Conference on Object-Oriented Programming*, Bologna, Italy, p. 139–149, 1994.

Bianchi, A.; Caivano, D.; Marengo, V.; Visaggio, G. Iterative reengineering of legacy systems. *IEEE Transactions on Software Engineering*, v. 29, n. 3, p. 225–241, 2003.

Binder, R. V. *Testing object-oriented systems: Models, patterns, and tools*. first ed. Addison-Wesley, 1999.

Bisbal, J.; Lawless, D.; Wu, B.; Grimson, J. Legacy information systems: issues and directions. *IEEE Software*, v. 16, n. 5, p. 103–111, 1999.

Black, S. Computing ripple effect for software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 2001, n. 13, p. 263–270, 2001.

Boehm, B. Software engineering. *IEEE Transactions on Computers*, v. 25, n. 12, p. 1226–1242, 1976.

Boehm, B. Get ready for agile methods, with care. *IEEE Computer*, v. 35, n. 1, p. 64–69, 2002.

Booch, G. *Object-oriented design*. Benjamim Cummings, 1991.

Bosch, J.; Molin, P.; Mattsson, M.; Bengtsson, P.; Fayad, M. E. *Building application frameworks: Object-oriented foundations of framework design*, cap. Framework Problems and Experiences. V. 1 de (Fayad et al., 1999), p. 55–82, 1999.

Braga, R. *Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico*. Tese de Doutoramento, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2003.

Braga, R. T. V. *GREN-Wizard User Guide*. Documento de trabalho, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, version 1.0, 2002a.

Braga, R. T. V. *Manual de instanciação do framework GREN*. Documento de trabalho, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2002b.

Braga, R. T. V.; Germano, F. S.; Masiero, P. C. Experiments on pattern language-based modeling. In: *SBES'2003, XVII Simpósio Brasileiro de Engenharia de Software*, Manaus, AM, p. 35–50, 2003.

Braga, R. T. V.; Germano, F. S. R.; Masiero, P. C. A pattern language for business resource management. In: *PLOP'1999, 6th Conference on Pattern Languages of Programs*, p. 1–33, 1999.

Braga, R. T. V.; Masiero, P. C. A process for framework construction based on a pattern language. In: *COMPSAC'2002, Annual International Computer Software and Applications Conference, 26*, Oxford, England: IEEE, p. 615–620, 2002.

Braga, R. T. V.; Masiero, P. C. Building a wizard for framework instantiation based on a pattern language. In: *OOIS'2003, International Conference on Object-Oriented Information Systems, 9*, Geneva, Switzerland: Lecture Notes on Computer Science, LNCS 2817, Springer, p. 95–106, 2003.

Brugali, D.; Menga, G.; Aarsten, A. A. *Domain-specific application frameworks: Frameworks experience by industry*, cap. A case study for flexible manufacuring systems. V. 1 de (Fayad e Johnson, 2000), p. 85–99, 2000.

Bäumer, D.; Gryczan, G.; Knoll, R.; Lilienthal, C.; Riehle, D.; Züllighoven, H. Framework development for large systems. *Communications of the ACM*, v. 40, n. 10, p. 52–59, 1997.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.; Sommerlad, P.; Stal, M. *Pattern-oriented software architecture: A system of patterns.* Wiley Series in Software Design Patterns, first ed. Wiley, 1996.

Cagnin, M. I. *Avaliação das vantagens quanto à facilidade de manutenção e expansão de sistemas legados sujeitos a engenharia reversa e segmentação.* Dissertação de mestrado, Departamento de Computação, UFSCar, São Carlos, 101 p., 1999.

Cagnin, M. I. *Estudo de Caso: Engenharia Reversa do Sistema de Biblioteca e Engenharia Avante utilizando o Framework GREN.* Documento de trabalho, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2002a.

Cagnin, M. I. *Estudo de Caso: Engenharia Reversa do Sistema de Oficina Eletrônica e Engenharia Avante utilizando o framework GREN.* Documento de trabalho, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2002b.

Cagnin, M. I.; Braga, R. T. V.; Germano, F. S.; Chan, A.; Maldonado, J. C. Extending Patterns with Testing Implementation. In: *SugarLoafPlop'2005, V Conferência Latino-Americana em Linguagens de Padrões para Programação*, Campos do Jordão, SP, (submetido), 2005a.

Cagnin, M. I.; Germano, F. S.; Penteado, R. D.; Maldonado, J. C. *Manual da Ferramenta GREN-WizardVersionControl.* Documento de trabalho, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, versão 1.0, 2004a.

Cagnin, M. I.; Maldonado, J. C.; Braga, R. T. V.; Germano, F. S.; Penteado, R. D. Uma Ferramenta de Apoio ao Controle de Versão das Aplicações Criadas por um Framework. In: *XXX Conferência Latino-Americana de Informática*, Arequipa, Peru, artigo selecionado para ser publicado no CLEI Electronic Journal, p. 414–425, 2004b.

Cagnin, M. I.; Maldonado, J. C.; Chan, A.; Penteado, R. D.; Germano, F. S. Reuso na Atividade de Teste para Reduzir Custo e Esforço de VV&T no Desenvolvimento e na Reengenharia de Software. In: *XVIII Simpósio Brasileiro de Engenharia de Software*, Brasília, DF, p. 71–85, 2004c.

Cagnin, M. I.; Maldonado, J. C.; Germano, F. S.; Chan, A.; Penteado, R. D. Um estudo de caso de reengenharia usando o processo PARFAIT. In: *SDMS'2003, Simpósio de Desenvolvimento e Manutenção de Software da Marinha*, Rio de Janeiro, RJ, CD-ROM, 10 p., 2003a.

Cagnin, M. I.; Maldonado, J. C.; Germano, F. S.; Masiero, P. C.; Penteado, R. D.; Braga, R. T.; Francelino, M. T. A Comprehensive Evolution Process for Application Frameworks. In: *Cadernos de Computação, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo*, São Carlos, SP, p. 71–93, 2004d.

Cagnin, M. I.; Maldonado, J. C.; Germano, F. S.; Masiero, P. C.; Penteado, R. D.; Chan, A. An agile reverse engineering process based on a framework. In: *WER'2003, 6th International Workshop on Requirements Engineering*, Piracicaba, SP, p. 240–254, 2003b.

Cagnin, M. I.; Maldonado, J. C.; Germano, F. S.; Penteado, R. D. PARFAIT: Towards a framework-based agile reengineering process. In: *ADC'2003, Agile Development Conference*, Salt Lake City, UTHA, USA: IEEE, p. 22–31, 2003c.

Cagnin, M. I.; Maldonado, J. C.; Germano, F. S.; Penteado, R. D.; Braga, R. T. GREN-WizardVersionControl: Uma Ferramenta de Apoio ao Controle de Versão das Aplicações Criadas pelo Framework GREN. In: *Sessão de Ferramentas'2004, Simpósio Brasileiro de Engenharia de Software*, Brasília, DF, p. 73–78, 2004e.

Cagnin, M. I.; Maldonado, J. C.; Masiero, P. C.; Penteado, R. D.; Braga, R. T. An Evolution Process for Application Frameworks. In: *I Workshop de Manutenção Moderna de Software, em conjunto com o XVIII Simpósio Brasileiro de Engenharia de Software*, Brasília, DF, CD-ROM, 8 p., 2004f.

Cagnin, M. I.; Maldonado, J. C.; Penteado, R.; Germano, F. PARFAIT: Definição e Exemplo de Aplicação, documento de Trabalho, Instituto de Ciências Matemáticas e de Computação-USP, 2004g.

Cagnin, M. I.; Penteado, R.; Masiero, P. C.; Braga, R. T. V.; Maldonado, J. C. Process for variability control and application frameworks evolution. *RTInfo – Revista Tecnologia da Informação*, v. 1, n. 5, (submetido), 2005b.

Carnegie Mellon University *Capability maturity model integration - version 1.1.* Relatório Técnico CMU/SEI-2002-TR-003, Carnegie Mellon University, Software Engineering Institute, 2002.

Cha, J.; Kim, C.; Yang, Y. Architecture based software reengineering approach for transforming from legacy system to component based system through applying design patterns. *Lecture Notes in Computer Science, Software Engineering Research and Applications: First International Conference, SERA 2003*, v. 3026/2004, p. 266–278, 2004.

Chan, A.; Cagnin, M. I.; Maldonado, J. C. Aplicação do Processo Ágil de Reengenharia PARFAIT em um Sistema Legado de Controle de Biblioteca, Documento de Trabalho, Instituto de Ciências Matemáticas e de Computação-USP, 2003.

Chapin, N.; Hale, J. E.; Khan, K. M.; Ramil, J.; Tan, W. Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 13, p. 3–30, 2001.

Chau, T.; Maurer, F.; Melnik, G. Knowledge Sharing: Agile Methods vs. Tayloristic Methods. In: *WETICE'2003, 12th IEEE International Workshop on Enabling Technologies: Infraestructure for Collaborative Enterprises*, p. 302–307, 2003.

Chessman, J.; Daniels, J. *Components – a simple process for specifying component-based software.* Addison-Wesley, 2001.

Chikofsky, J. E.; Cross, J. H. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, v. 7, n. 1, p. 13–17, 1990.

Christie, M. A. *Process-centered development environments: An exploration of issues.* Technical Report CMU/SEI-93-TR-4, SEI, Software Engineering Institute, Carnigie Mellon University, Pittsburgh, Pennsylvania, 1993.

Chu, W. C.; Lu, C.; Chang, C.; Chung, Y. *Handbook of software engineering and knowledge engineering*, cap. Pattern-based Software Re-engineering. first ed World Scientific Publishing Company, 2001.

Chu, W. C.; Lu, C.; Shiu, C.; He, X. Pattern-based software reengineering: a case study. *Journal of Software Maintenance: Research and Practice*, v. 12, p. 121–141, 2000.

Cimitile, A.; Lucia, A. D.; Lucca, G. A. D.; Fasolino, A. R. Identifying objects in legacy systems using design metrics. *The Journal of Systems and Software*, v. 44, n. 3, p. 199–211, 1999.

Cincom Visualworks 5i.4 non-commercial. `http://www.cincom.com/index.html`. Acesso em: Fevereiro/2003, 2003.

Clauss, M. Modeling Variability with UML. In: *GCSE'2001, 3rd International Symposium on Generative and Component-Based Software Engineering, Young Researchers Workshop*, Erfurt, Germany, p. 226–230, 2001.

CLEMENTS, P.; NORTHROP, L. *Software product lines: Practices and patterns.* SEI Series in Software Engineering. Addison-Wesley, 2001.

COAD, P. Object-oriented patterns. *Communications of the ACM*, v. 35, n. 9, p. 152–159, 1992.

COAD, P. *Java Modeling in Color with UML.* Prentice-Hall, 1999.

COAD, P.; NORTH, D.; MAYFIELD, M. *Object models: Strategies, patterns and applications.* second ed. Yourdon Press, 1997.

COAD, P.; YOURDON, E. *Object-oriented analisys.* second ed. Prentice-Hall, 1991.

COCKBURN, A.; HIGHSMITH, J. Agile software development: The people factor. *IEEE Computer*, v. 34, n. 11, p. 131–133, 2001.

COHN, M.; FORD, D. Introducing an Agile Process to an Organization. *IEEE Computer*, v. 36, n. 6, p. 74–78, 2003.

CONCURRENT VERSIONS SYSTEM Concurrent versions system - the open standard for version control. `http://www.cvshome.org`, acesso em Abril/2004, 2004.

COPLIEN, J. Organizational patterns. site, `http://www.easycomp.org/cgi-bin/OrgPatterns?CoplienOrganizationPatternsIntroduction`. Acesso em Maio/2005., 1997.

COPLIEN, J.; HARRISON, N. *Organizational patterns for agile software development.* Prentice Hall, 2004.

COPLIEN, J. O. *The patterns handbook: Techniques, strategies, and applications*, cap. Software Design Patterns: Common Questions and Answers Cambridge University Press, p. 311–320, 1998.

COSTA, R. M. *Fusion-RE/I: Um Método de Engenharia Reversa para Apoiar Manutenção do Software.* Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos-SP, 112 p., 1997.

CRISPIN, L.; HOUSE, T. *Testing extreme programming.* The XP Series. Addison-Wesley, 2003.

CRISPIN, L.; HOUSE, T.; WADE, C. The need for speed: Automating acceptance testing in an extreme programming environment. In: *XP'2001, 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, 2001.

CUSUMANO, M. A.; YOFFIE, D. B. Software development on Internet Time. *IEEE Computer*, v. 32, n. 10, p. 60–69, 1999.

D. Coleman et al *Object-oriented development - the fusion method.* Prentice-Hall, 1994.

Dallal, J. A.; Sorenson, P. System testing for object-oriented frameworks using hook technology. In: *ASE'2002, 17th IEEE International Conference on Automated Software Engineering*, Edinburgh, UK, p. 231–236, 2002.

Deelstra, S.; Sinnema, M.; Bosch, J. Product derivation in software product families: A case study. *Journal of Systems and Software*, v. 74, n. 2, p. 173–194, 2005.

Deelstra, S.; Sinnema, M.; Nijhuis, J.; Bosch, J. COSVAM: A Technique for Assessing Software Variability in Software Product Families. In: *ICSM'2004, 20th IEEE International Conference on Software Maintenance*, Chicago Illinois, USA: IEEE, p. 458–462, 2004.

DeLano, D. E.; Rising, L. *Pattern languages of program design 3*, cap. Software Design Patterns: Common Questions and Answers. V. 1 de (Martin et al., 1998), p. 503–525, 1998.

DeMarco, T. *Structured analysis and system specification.* Prentice-Hall, 1979.

Demeyer, S.; Ducasse, S.; Nierstrasz, O. A pattern language for reverse engineering. In: *EuroPLOP'2000, 5th European Conference on Pattern Languages of Programming and Computing*, Irsee, Germany: Andreas Ruping, p. 189–208, 2000.

van Deursen, A.; Kuipers, T. *Object identification using cluster and concept analysis.* Technical Report SEN-R9814, Centrum voor Wiskunde en Informatica, `http://ftp.cwi.nl/CWIreports/SEN/SEN-R9814.pdf`. Acesso em Fevereiro/2002, 1998.

Esprit Project 29512 *Renaissance method revision.* Technical Report D.3.2.1, Lancaster University, `http://www.comp.lancs.ac.uk/computing/research/cseg/projects/emergency/downloadings/emgD321_final.doc`, 1999.

Fayad, M. E.; Hamza, H. The AnyAccount Pattern. In: *PLOP'2003, 10th Conference on Pattern Languages of Programs 2003*, Monticello, Illinois, 2003.

Fayad, M. E.; Johnson, R. E. *Domain-specific application frameworks: Frameworks experience by industry.* first ed. John Wiley & Sons, 2000.

Fayad, M. E.; Johnson, R. E.; Schmidt, D. C. *Building application frameworks: Object-oriented foundations of framework design.* first ed. John Wiley & Sons, 1999.

Fayad, M. E.; Schmidt, D. C. Object-oriented application frameworks. *Communications of the ACM*, v. 40, n. 10, p. 32–38, 1997.

FERNANDEZ, E. B.; LIU, Y. The account analysis pattern. In: *EuroPLoP'2002, 7th European Conference on Pattern Languages of Programs*, Irsee, Germany, 2002.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. *Refactoring: Improving the design of existing code.* first ed. Addison-Wesley, 1999.

FOWLER, M.; SCOTT, K. *UML Distilled - Applying the Standard Object Modeling Language.* first ed. Addison-Wesley, 1997.

FRANCH, X.; RIBÓ, J. M. Using UML for modelling the static part of a software process. In: *UML'1999, 2nd International Conference on the Unified Modeling Language, Beyond the Standard*, Fort Collins CO, USA: Springer-Verlag, p. 292–307, 1999.

FROEHLICH, G.; HOOVER, J.; LIU, L.; SORENSON, P. Hooking into object-oriented application frameworks. In: *19th International Conference on Software Engineering*, Boston, USA, p. 491–501, 1997.

FUGGETA, A. Software Process: a Roadmap. In: *22nd International Conference on the Future of Software Engineering*, Limerick, Ireland: ACM–Association for Computing Machinery, p. 25–34, 2000.

GALL, H.; KLÖSCH, R. Capsule oriented reverse engineering for software reuse. In: *ESEC'1993, 4th European Software Engineering Conference*, Garmish-Partenkirchen, Germany, p. 418–433, 1993.

GALL, H.; KLÖSCH, R. Finding objects in procedural programs: An alternative approach. In: *WCRE'1995, 2nd Working Conference on Reverse Engineering*, Toronto, Canada, p. 208–216, 1995.

GALL, H.; KLÖSCH, R.; MITTERMEIR, R. Application patterns in re-engineering: Identifying and using reusable concepts. In: *IPMU'1996, 6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Granada, Spain, p. 1099–1106, 1996.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design patterns elements of reusable of object-oriented software.* second ed. Addison-Wesley, 1995.

GANE, T.; SARSON, C. *Structured systems analysis.* McDonnell Douglas, 1982.

GENVIGIR, E. C.; SANT'ANNA, N.; FILHO, L. F. B.; JUNIOR, M. G. C.; CASILLO, B. H. Modelando Processos de Software através do UPM - Modelo de Processo Unificado. In: *CITS'2003, Congresso Internacional de Tecnologia de Software*, Curitiba, Paraná, p. 28–41, 2003.

GIMENES, I. M. S.; HUZITA, E. H. M. *Desenvolvimento baseado em componentes: Conceitos e técnicas.* primeira ed. Editora Ciência Moderna, 2005.

GIORGINI, P.; KOLP, M.; MYLOPOULOS, J. Organizational patterns for early requirement analysis. In: *RE'2002, 10th IEEE Joint International Requirements Engineering Conference*, Essen, Germany, 2002.

GOEDICKE, M.; ZDUN, U. Piecemeal legacy migrating with an architectural pattern language: a case study. *Journal of Software Maintenance Aand Evolution: Research and Practice*, v. 14, n. 1, p. 1–30, 2002.

HAREL, D. Statecharts: A visual formalism to complex systems. *Science of Computer Programming*, v. 8, p. 231–274, 1987.

HARROLD, M. J. Testing: a roadmap. In: *ICSE'2000, 22nd International Conference on Software Engineering, The Future of Software Engineering*, New York, NY, USA: ACM Press, p. 61–72, 2000.

HAYES, J. H.; OFFUTT, A. J. Increased software reliability through input validation analysis and testing. In: *ISSRE'1999, 10th International Symposium on Software Reliability Engineering*, Boca Raton, FL, USA, p. 199–209, 1999.

HIGHSMITH, J.; COCKBURN, A. Agile Software Development: The Business of Inovation. *IEEE Computer*, v. 34, n. 9, p. 120–122, 2001.

HIRSCHFELD, R. Aspects homepage. site, `http://www-ia.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/AspectS.html`. Acesso em Janeiro/2003, 2003.

HUMPHREY, S. W.; KELLNER, M. *Software Process Modeling: Principles of Entity Process Models.* Technical Report CMU/SEI-89-TR-2, SEI, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1989.

HUNT, A.; THOMAS, D. *The pragmatic programmer: From journeyman to master.* first ed. Addison-Wesley, 1999.

IBM RATIONAL SOFTWARE `http://www-306.ibm.com/software/rational/`. Acesso em Dezembro/2003, 2003.

IEEE IEEE Standard Glossary of Software Engineering Terminology, Standard 610.12. 1990.

ISO Iso 9001, quality systems: Model for quality assurance in: Design, development, production, instalation and servicing. 1994.

ISO ISO/IEC 12207:1995, Information Technology - Software Life Cycle Processes. International Standard Organization, 1995.

ISO Iso/iec tr 15504: Information technology – software process assessment. 1998.

ISO Iso/iec 14598-1, information technology – software product evaluation – part 1: General overview. 1999a.

ISO Iso/iec 9126-1, information technology – software product quality – part 1: Quality model. 1999b.

J. Siegel and the OMG Staff Strategy Group Developing in OMG's Model-Driven Architecture. White Paper, 2001.

Jaccheri, M. L.; Picco, G. P.; Lago, P. Eliciting software process models with E3 language. *ACM Transactions on Software Engineering and Methodology*, v. 7, n. 4, p. 368–410, 1998.

Jackson, M. A. *System development.* Prentice-Hall, 1983.

Jacobson, I. Re-engineering of old systems to an object-oriented architecture. In: *OOPSLA'01, ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Tampa Bay, Florida, USA, p. 340–350, 1991.

Jacobson, I. *Object-oriented software engineering: a use case driven approach.* Reading, MA: Addison-Wesley, 1995.

Jacobson, I.; Jacobson, S. Reengineering your software engineering process. *Object Magazine*, v. 4, n. 2, 1995.

Jeffries, R. Extreme Testing - Why Aggressive Software Development Calls for Radical Testing Efforts. *Software Testing & Quality Engineering*, p. 23–26, 1999.

Jeon, T.; Seung, H. W.; Lee, S. Embedding built-in tests in hot spots of an object-oriented framework. *ACM SIGPLAN Notices*, v. 37, n. 8, p. 25–34, 2002.

Johnson, R. E. Documenting Frameworks using Patterns. In: *OOPSLA'1992, ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Seattle, USA, p. 63–76, 1992.

Johnson, R. E.; Foote, B. Designing reusable classes. *Journal of Object Oriented Programming – JOOP*, v. 1, n. 2, p. 22–35, 1988.

Kellner, M.; Hansen, A. G. *Software process modeling.* Technical Report CMU/SEI-88-TR-9, SEI, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1988.

Kiczales, G.; Irwin, J.; Lamping, J.; Loingtier, J.-M.; Lopes, C.; Maeda, C.; Menhdhekar, A. Aspect-oriented programming. In: Akşit, M.; Matsuoka, S., eds. *ECOOP'97, 11th European Conference on Object-Oriented Programming*, Berlin, Heidelberg, and New York: Springer-Verlag, p. 220–242, 1997.

Kruchten, P. *The Rational Unified Process: An Introduction.* Second ed. Addison-Wesley, 298 p., 2000.

K.Schwaber; Beedle, M. *SCRUM: Agile Software Development.* Prentice-Hall, 2002.

Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* third ed. Prentice Hall, 2004a.

Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, cap. Test-Driven Development and Refactoring. third ed Prentice Hall, p. 385–394, 2004b.

Larsen, G. Designing Component-Based Frameworks Using Patterns in the UML. *Communications of the ACM*, v. 42, n. 10, p. 38–45, 1999.

Laudon, K. C.; Laudon, J. P. *Management information systems - managing the digital firm.* seventh ed. Prentice Hall, 2002.

Lemos, G. S. *PRE/OO - Um Processo de Reengenharia Orientada a Objetos com Ênfase em Garantia de Qualidade.* Dissertação de mestrado, Departamento de Computação. Universidade Federal de São Carlos, São Carlos-SP, 159 p., 2002.

M., N. J. A.; Moreno *Lecture notes on empirical software engineering.* Series on Software Engineering and Knowledge Engineering, first ed. World Scientific Publishing Company, 2003.

Maldonado, J. C.; Fabbri, S. C. P. F. *Qualidade de software: Teoria e prática*, cap. Verificação e Validação de Software. In: (Rocha et al., 2001), p. 66–73, 2001a.

Maldonado, J. C.; Fabbri, S. C. P. F. *Qualidade de software: Teoria e prática*, cap. Teste de Software. V. 1 de (Rocha et al., 2001), p. 73–84, 2001b.

Markiewicz, M.; Lucena, C. Object oriented framework development. *ACM Crossroads Student Magazine*, crossroads 7.4, Summer 2001. `http://acm.org/crossroads/xrds7-4/frameworks.html`. Acesso em Janeiro/2005, 2001.

Martin, R. C.; Riehle, D.; Buschmann, F. *Pattern languages of program design 3.* first ed. Addison-Wesley, 1998.

Masiero, P. C.; Fortes, R. P. M.; Batista, J. E. S. Edição e simulação de aspectos comportamentais de sistemas de tempo real. In: *Seminário Integrado de Software e Hardware, Anais do Congresso da Sociedade Brasileiro de Computação*, p. 45–61, 1991.

Maurer, F.; Martel, S. Extreme Programming: Rapid Development for Web-Based Applications. *IEEE Internet Computing*, v. 6, n. 1, p. 86–90, 2002.

McConnell, S. *Code complete.* first ed. Microsoft Press, 1993.

McMenamim, S.; Palmer, J. *Análise Essencial de Sistemas.* McGraw-Hill, 1991.

MCT Caracterização do software. *Qualidade e Produtividade no Setor de Software Brasileiro, Ministério da Ciência e Tecnologia, Secretaria de Política de Informática*, , n. 4, p. 37–40, `http://www.mct.gov.br/sepin/Dsi/Software/Menu_Qualidade.htm`. Acesso em Maio/2005, 2002.

MERANT MicroFocus Products - Revolve. `http://www.merant.com/products/microfocus/revolve/`, 2000.

Miller, G. What's New in UML 2.0? Part I. A Borland White Paper, `http://bdn.borland.com/article/images/31881/Together_White_paper__.pdf`. Acesso em Janeiro/2005, 2003.

Miller, J.; Mukerji, J. *Model Driven Architecute (MDA).* Relatório Técnico ormsc/2001-07-01, Object Management Group (OMG), Architecture Board ORMSC, 2001.

Monday, P.; Carey, J.; Dangler, M. *SanFrancisco Component Framework: An Introduction.* Addison-Wesley, 2000.

Myers, G. J. *The art of software testing.* second ed. Wiley, 2004a.

Myers, G. J. *The art of software testing*, cap. Extreme Testing. In: (Myers, 2004a), p. 177–191, 2004b.

MySQL MySQL Reference Manual. `http://www.mysql.com/doc/en/index.html`. Acesso em Dezembro/2003, 2003.

Nakagawa, E. Y. *Software Livre: Processo e Produto Livres no Desenvolvimento de Aplicações Web.* Exame de Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2002.

OMG Unified Modeling Language Specification. Version 1.5, formal/2003-03-01. `http://www.omg.org/technology/documents/formal/uml.htm`. Acesso em Abril/2005, 2003.

OMG Software process engineering metamodel specification. Object Management Group, Version 1.1, formal/05-01-06, 2005.

Omnisphere, I. S. C. OmniBuilder - Open Application Generator. site, `http://www.omni-sphere.com/overview/overview.htm`. Acesso em Fevereiro/2002, 2002.

Paulk, M. C. *Capability maturity model for software.* Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnigie Mellon University, Pittsburgh PA, version 1.1, 1993.

Pazin, A. *GAwCRe: Um Gerador de Aplicações baseadas na Web para o Domínio de Clínicas de Reabilitação.* Dissertação de mestrado, Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2004.

Pazin, A.; Penteado, R.; Masiero, P. C. SiGCli: A Pattern Language for Rehabilitation Clinics Management. In: *SugarLoafPlop'2004, IV Conferência Latino-Americana em Linguagens de Padrões para Programação*, Porto das Dunas-CE, p. 1–25, 2004.

Penteado, R. D. *Um Método para Engenharia Reversa Orientada a Objetos.* Tese de doutorado, Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos-SP, 237 p., 1996.

Perens, B. *Open sources: Voices of the open source revolution*, cap. The Open Source Definition. first ed O'Reilly & Associantes, p. 171–188, 1999.

Pfleeger, S. L.; Bohner, S. A. A framework for software maintenance metrics. In: *ICSM'1999, International Conference on Software Maintenance*, Oxford, England, UK, p. 320–327, 1999.

Polo, M.; Piattini, M.; Ruiz, F.; Calero, C. MANTEMA: A complete rigorous methodology for supporting maintenance based on the ISO/IEC 12207 standard. In: *CSMR'1999, 3rd European Conference on Software Maintenance and Reengineering*, Amsterdam, The Netherlands, p. 178–181, 1999.

Postema, M.; Schmidt, H. W. Reverse engineering and abstraction of legacy systems. *Informatica: An International Journal of Computing and Informatics*, v. 22, n. 3, p. 359–371, 1998.

Pree, W. *Design patterns for object oriented software development.* Workingham: Addison-Wesley, 1995.

Pree, W. *Building application frameworks: Object-oriented foundations of framework design*, cap. Hot-spot-driven development. V. 1 de (Fayad et al., 1999), p. 379–393, 1999.

Pressman, R. *Software engineering: A practitioner's approach.* sixth ed. McGraw-Hill, 2005.

Ré, R. *Um Processo para Construção de Frameworks a partir da Engenharia Reversa de Sistemas de Informação baseados na Web: Aplicação do Domínio dos Leilões Virtuais.* Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20052003-120738/publico/dissert-f.pdf`. Acesso em Janeiro/2005, 2002.

Ré, R.; Braga, R. T. V.; Masiero, P. C. A Pattern Language for Online Auctions Management. In: *PLOP'2001, 8th Conference on Pattern Languages of Programs*, Monticello, IL, EUA, p. 1–18, 2001.

R. Soley and the OMG Staff Strategy Group Model driven architecture. White Paper, 2000.

Rahgozar, M.; Oroumchian, F. An effective strategy for legacy system evolution. *Journal of Software Maintenance and Evolution Research and Practice*, v. 15, n. 5, p. 325–344, 2003.

Rajlich, V. A. Methodology for incremental changes. In: *XP'2001, 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, Sardinia, Italy, p. 20–231, 2001.

Ramesh, S. Application scenario: A pattern language for business process control. In: *PLOP'1998, 5th Conference on Pattern Languages of Programs*, Monticello, Illinois, USA, p. 1–19, 1998.

Recchia, E. L. *Engenharia reversa e reengenharia baseadas em padrões.* Dissertação de mestrado, Departamento de Computação. Universidade Federal de São Carlos, São Carlos-SP, 159 p., 2002.

Reifer, D. How good are agile methods? *IEEE Software*, v. 19, n. 4, p. 16–18, 2002.

Roberts, D.; Johnson, R. *Pattern languages of program design 3*, cap. Evolving frameworks: A pattern language for developing object oriented frameworks. V. 1 de (Martin et al., 1998), p. 471–486, 1998.

Rocha, A. D.; Simão, A. S.; Maldonado, J. C.; Masiero, P. C. Teste funcional: Uma abordagem auxiliada por aspectos. In: *I Workshop Brasileiro em Desenvolvimento de Software Orientado a Aspectos, em conjunto com o XVIII Simpósio Brasileiro de Engenharia de Software*, Brasília-DF, p. 1–8, 2004.

Rocha, A. R.; Maldonado, J.; Weber, K. *Qualidade de software: Teoria e prática.* primeira ed. Prentice Hall, 2001.

ROPER, M. *Software testing.* The International Software Engineering Series. McGraw-Hill, 1994.

ROSENBERG, L. H. *Software re-engineering.* Technical Report SATC-TR-95-1001, NASA, `http://satc.gsfc.nasa.gov/support/reengrpt.PDF`. Acesso em Dezembro/2002, 1996.

SCHMIDT, D.; BUSCHMANN, F. Patterns, frameworks, and middleware: their synergistic relationships. In: *ICSE'03, 25th International Conference on Software Engineering*, Portland, Oregon, p. 694–704, 2003.

SELIC, B. Tutorial: An Overview of UML 2.0. In: *26th International Conference on Software Engineering (ICSE'04)*, Scotland, UK, p. 741–742, 2004.

SHAFER, D. *Practical Smalltalk: using Smalltalk.* Springer-Verlag, 1991.

SHULL, F.; LANUBILE, F.; BASILI, V. Investigating reading techniques for object-oriented framework learning. *IEEE Transactions on Software Engineering*, v. 26, n. 11, p. 1101–1118, 2000.

SILVA, F. O. *Extensões ao Framework GREN Visando a Melhoria de Desempenho das Aplicações Instanciadas.* Relatório técnico bolsa pibic, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2004a.

SILVA, M. T. F. *Uso de aspectos para apoiar o desenvolvimento de frameworks: aplicação ao framework gren.* Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos-SP, 2004b.

SINNEMA, M.; DEELSTRA, S.; NIJHUIS, J.; BOSCH, J. Managing variability in software product families. In: *2nd Groningen Workshop on Software Variability Management*, Groningen, The Netherlands, 2004a.

SINNEMA, M.; DEELSTRA, S.; NIJHUIS, J. A. G.; BOSCH, J. COVAMOF: A Framework for Modeling Variability in Software Product Families. In: *SBLC'2004, 3rd Software Product Line Conference*, Boston, USA: Springer Verlag Lecture Notes on Computer Science, p. 197–213, 2004b.

SNEED, H. M. Planning the reengineering of legacy system. *IEEE Software*, v. 12, n. 1, p. 24–34, 1995.

SNEED, H. M. Architecture and functions of a commercial software reengineering workbench. In: *CSMR'1998, 2nd Euromicro Conference on Software Maintenance and Reengineering*, Florence, Italy, p. 2–10, 1998.

Sneed, H. M.; Nyáry, E. Extracting object-oriented specification from procedurally oriented programs. In: *WCRE'1995, 2nd Working Conference on Reverse Engineering*, Toronto-Canada, p. 217–226, 1995.

Soares, M. D.; Fortes, R. P. M.; Moreira, D. A. VersionWeb: A Tool for Helping Web Pages Version Control. In: *IMSA'2000, 4th International Conference on Internet Multimedia Systems and Applications*, Las Vegas, Nevada, USA, p. 275–280, 2000.

Sommerville, I. *Software engineering.* 6th ed. Addison-Wesley, 2000.

Sorgente, T.; Fernandez, E. B.; Petrie, M. L. Analysis patterns for patient treatment. In: *PLOP'2004, 11th Conference on Pattern Languages of Programs*, Monticello, Illinois, p. 1–9, 2004.

Stapleton, J. *Dsdm dynamic systems development method: The method in practice.* Addison-Wesley, 1997.

Stevens, P.; Pooley, R. Systems reengineering patterns. In: *6th Internatinal Symposium on the Foundations of Software Engineering*, Orlando, Florida, USA, p. 17–23, 1998.

Tahvildari, L.; Kontogiannis, K. On the role of design patterns in quality-driven re-engineering. In: *CSMR'2002, 6th European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, p. 230–240, 2002.

Tahvildari, L.; Kontogiannis, K.; Mylopoulos, J. Quality-driven software re-engineering. *The Journal of Systems and Software*, v. 2003, n. 66, p. 225–239, 2003.

Taligent Building object-oriented frameworks. `http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf`. Acesso em Fevereiro/2002, 1997.

Taligent Leveraging object-oriented frameworks. `http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/leveragingoo.pdf`. Acesso em Fevereiro/2002, 1998.

Taschwer, M.; Rauner-Reithmayer, D.; Mittermeir, R. Generating Objects from C Code - Features of the CORET Tool-Set. In: *CSMR'1999, 3rd European Conference on Software Maintenance and Reengineering*, Amsterdan, The Netherlands, p. 91–101, 1999.

Terekhov, A.; Verhof, C. The realities of language conversion. *IEEE Software*, v. 17, n. 6, p. 111–124, 2000.

Tilley, S. *Perspectives on legacy system reengineering.* Technical report, Software Engineering Institute, Carnigie Mellon University, Pittsburgh, PA, `http://www.sei.cmu.edu/~reengineering/pubs/lsysree/`. Acesso em Dezembro/2002, 1995.

TOURWÉ, T. *Automated support for framework-based software evolution.* Tese de Doutoramento, Department of Computer Science, Vrije Universiteit Brussel, Brussel, 2002.

TSAI, W.; TU, Y.; SHAO, W.; EBNER, E. Testing extensible design patterns in object-oriented frameworks through scenario templates. In: *COMPSAC'1999, 23rd International Computer Software and Applications Conference*, Phoenix, AZ, p. 166–171, 1999.

TURK, D.; FRANCE, R.; RUMPE, B. Limitations of agile software processes. In: *Third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2002)*, Alghero, Sardinia, Italy, p. 43–46, 2002.

ULRICH, W. M. The Evolutionary Growth of Software Reengineering and the Decade Ahead. *American Programmer*, v. 3, n. 10, p. 14–20, 1990.

WARREN, I.; RANSOM, J. Renaissance: A Method to Support Software System Evolution. In: *COMPSAC'2002, 26th Annual International Computer Software and Applications Conference*, Oxford, England, p. 415–420, 2002.

WEBER, K. C.; ROCHA, A. R.; ALVES, A.; AYALA, A. M.; GONÇALVES, A.; PARET, B.; SALVIANO, C.; MACHADO, C. F.; SCALET, D.; PETIT, D.; ARAÚJO, E.; BARROSO, M. G.; OLIVEIRA, K.; OLIVEIRA, L. C. A.; AMARAL, M. P.; CAMPELO, R. E. C.; MACIEL, T. Modelo de referência para melhoria de processo de software: Uma abordagem brasileira. In: *XXX Conferência Latino-Americana de Informática*, Arequipa, Peru, p. 1–16, 2004.

WEIDERMAN, N. H.; BERGEY, J. K.; SMITH, D. B. *Approaches to legacy system evolution.* Technical Report CMU/SEI-97-TR-014, Software Engineering Institute, Carnigie Mellon University, Pittsburgh PA, 1997.

WEISS, D. M.; LAI, C. T. R. *Software product-line engineering: a family-based software development process.* Addison-Wesley, 1999.

WHOLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering: An introduction.* Kluwer, 2000.

WILLIAMS, L.; COCKBURN, A. Agile software development: it's about feedback and change. *IEEE Computer*, v. 36, n. 6, p. 39–43, 2003.

YASSIN, A.; FAYAD, M. E. *Domain-specific application frameworks: Frameworks experience by industry*, cap. Application Frameworks: A Survey. V. 1 de (Fayad e Johnson, 2000), p. 615–632, 2000.

YAU, S.; COLLOFELLO, J. S. Some stability measures for software maintenance. *IEEE Transactions on Software Engineering*, v. 6, n. 6, p. 545–552, 1980.

YODER, J. M.; JOHNSON, R. E.; WILSON, Q. D. Connecting business objects to relational databases. In: *PLOP'1998, 5th Conference on the Pattern Languages of Programs*, Monticello-IL, EUA, `http://citeseer.ist.psu.edu/yoder98connecting.html`. Acesso em Abril/2002, 1998.

YOURDON, E. N. *Modern structured analysis.* Prentice-Hall, 1989.

YOURDON, E. N.; CONSTANTINE, L. L. *Structured design.* Yourdon Press, 1978.

ZHAO, J.; YANG, H.; XIANG, L.; XU, B. Change impact analysis to support architectural evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 2002, n. 14, p. 317–333, 2002.

ZOU, Y.; KONTOGIANNIS, K. Migration to object oriented platforms: A state transformation approach. In: *ICSM'2002, 18th International Conference on Software Maintenance*, Montreal, Quebec, Canada, p. 530–539, 2002.

Apêndice

A

Pacote de Experimentação

Nesta seção apresenta-se um pacote de experimentação no contexto de reengenharia, que tem como principal objetivo analisar o processo de reengenharia baseado em framework, proposto no Capítulo 3, e colaborar para a sua evolução, já que com o apoio deste pacote experimentos controlados podem ser repetidos tanto no ambiente acadêmico quanto industrial. O planejamento e a definição desse pacote foram parcialmente validados com o apoio da execução de um estudo de caso, conduzido no ambiente acadêmico, e está relatado na Seção A.0.3.

A definição do pacote é baseada na proposta de Wholin et al. (2000), que é dividida em cinco fases: definição, planejamento, operação, análise e interpretação e, apresentação e empacotamento. No entanto, apenas as três primeiras fases são descritas nas subseções a seguir, pois apesar da autora ter tido iniciativa de realizar um estudo de caso para avaliar o planejamento do experimento do pacote, o mesmo não foi concluído com sucesso, conforme justificativa apresentada na Subseção A.0.3. Assim, nesta seção comentam-se sobre a fase de preparação do estudo de caso (ou seja, preparar e organizar o material, escolher os participantes, informar os participantes sobre o objetivo do experimento, obter o consentimento deles em participar do experimento, informar sobre o sigilo, conduzir os treinamentos necessários e distribuir o material aos participantes) e sobre os problemas encontrados durante a sua execução e durante a validação dos dados coletados. Ressalta-se que a última fase do pacote de experimentação (apresentação e empacotamento) também foi parcialmente executada pois o pacote foi alimentado com informações obtidas durante as duas fases iniciais do pacote de experimentação (definição e planejamento) e exercitadas durante a operação do estudo de caso.

A.0.1 Fase de Definição

Analisar o PARFAIT, um processo ágil de reengenharia baseado em framework.

Para o propósito de avaliação da aplicabilidade do PARFAIT.

Com respeito à efetividade de apoio do PARFAIT na reengenharia de sistemas procedimentais para OO.

Do ponto de vista da reengenharia de sistemas procedimentais para o paradigma OO.

No contexto de alunos e profissionais interessados na reengenharia de software no domínio de Gestão de Recursos de Negócios, usando o PARFAIT e outro processo qualquer de reengenharia "ad hoc" em dois sistemas legados: um de controle de oficina eletrônica, desenvolvido em Clipper com aproximadamente 4,5 KLOC, e um de controle de biblioteca, também desenvolvido em Clipper, com aproximadamente 6 KLOC. Os participantes do experimento são selecionados de acordo com a sua disponibilidade e interesse. O perfil de cada participante (por exemplo, experiência profissional, experiência no objeto de estudo, experiência na linguagem de programação fonte e alvo, nível de conhecimento no domínio, entre outros) é obtido com a aplicação de um questionário, disponível no Apêndice C.

Na Tabela A.1 é apresentado um resumo da fase de definição do pacote de experimentação.

Tabela A.1: Resumo da fase de definição do pacote de experimentação

Objeto de estudo	**Propósito**	**Enfoque de qualidade**	**Perspectiva**	**Contexto**
Processo de reengenharia PARFAIT.	Avaliar.	Efetividade.	Reengenharia de sistemas procedimentais para OO.	Alunos e profissionais interessados na reengenharia de sistemas procedimentais para o paradigma OO no domínio de Gestão de Recursos de Negócios.

A.0.2 Fase de Planejamento

a) **Seleção do contexto:** O experimento pode ser conduzido por profissionais e/ou alunos interessados na reengenharia de sistemas procedimentais para o paradigma OO com sistemas legados reais, em uso atualmente, pertencentes a um domínio específico (ou seja, Gestão de Recursos de Negócios).

b) **Formulação das hipóteses:** As hipóteses do pacote foram definidas com o intuito de analisar o reúso, a qualidade do sistema resultante e a redução do tempo e, conseqüentemente, do custo da reengenharia com o apoio do PARFAIT.

- **Hipóteses Nulas:**

  - H0-1: O nível de reúso de software nas fases de análise, implementação e teste com o uso do processo PARFAIT é menor do que com o uso de processos de reengenharia "ad hoc".
  - H0-2: A qualidade do novo sistema resultante do processo PARFAIT é menor do que aquele resultante de processos de reengenharia "ad hoc".
  - H0-3: O tempo de reengenharia quando se usa o processo PARFAIT é maior do que quando se usam processos de reengenharia "ad hoc".

- **Hipóteses Alternativas:**
  - HA-1: O nível de reúso de software nas fases de análise, implementação e teste com o uso do processo PARFAIT é maior do que com o uso de processos de reengenharia "ad hoc".
  - HA-2: A qualidade do novo sistema resultante do processo PARFAIT é maior do que aquele resultante de processos de reengenharia "ad hoc".
  - HA-3: O tempo de reengenharia quando se usa o processo PARFAIT é menor do que quando se usam processos de reengenharia "ad hoc".

c) Seleção de Variáveis: A seguir, são listadas as variáveis independentes e as variáveis dependentes que devem ser consideradas no experimento.

- **Variáveis independentes:** *Experiência dos participantes:* no paradigma OO, na UML, na linguagem de padrões GRN, no framework GREN, na linguagem de programação Smalltalk, na linguagem de programação Clipper, no domínio de Gestão de Recursos de Negócios, nos critérios de teste funcional Particionamento de Equivalência e Análise do Valor Limite e nas diretrizes para reúso de teste da abordagem ARTe (Capítulo 5). *Metodologia utilizada:* Na primeira etapa, um processo de reengenharia "ad hoc" é utilizado, sendo que metade dos grupos/participantes faz a reengenharia de um sistema legado de biblioteca e a outra metade faz a reengenharia em um sistema legado de oficina eletrônica, ambos desenvolvidos em linguagem de programação Clipper. Na segunda etapa, o processo PARFAIT é utilizado na reengenharia, invertendo-se o sistema legado entre os grupos/participantes;
- **Variáveis dependentes:** quantidade total de funcionalidades do sistema legado, quantidade total de funcionalidades modeladas a partir de reúso, quantidade total de funcionalidades implementadas a partir de reúso, quantidade total de casos de teste criados para validar o novo sistema, quantidade total de casos de teste reutilizados para validar o novo sistema, quantidade de erros encontrados pelos usuários durante a demonstração do novo sistema, quantidade de tarefas que o usuário consegue executar a partir de um conjunto de tarefas determinadas,

quantidade de tarefas que o usuário deseja mas não consegue executar, tempo despendido para conduzir a reengenharia.

d) Seleção dos Participantes: Os participantes do experimento são selecionados por amostragem não-probabilística por conveniência entre profissionais ou alunos interessados na reengenharia de sistemas.

e) Restrições e Limitações: Pode haver restrições de tempo e custo para a execução do experimento, pois os participantes são profissionais/alunos. Além disso, pode haver limitação com respeito a equipamentos (quantidade e configuração de computadores) necessários para a condução do experimento.

f) Projeto do experimento: O experimento pode ser realizado individualmente ou em grupo. Esse último quando há interesse de analisar a cooperação na reengenharia. Para a seleção dos componentes de cada grupo, sugere-se que a experiência de um participante complemente a dos outros, para que o nível de conhecimentos dos grupos seja equivalente. O projeto do experimento é em bloco, sendo que haverá troca dos processos de reengenharia e também dos sistemas legados entre os indivíduos ou grupos, se for o caso, conforme apresentado na Tabela A.2. O projeto é do tipo um fator com dois tratamentos, ou seja: **Fator:** Reengenharia de Sistemas Procedimentais para o Paradigma OO. **Tratamentos:** a) utilizando processo "ad hoc" e b) utilizando processo PARFAIT.

Tabela A.2: Projeto do experimento

Fator: Reengenharia de Sistemas Procedimentais para OO			
Tratamento 1: Utilizando processo "ad hoc"		Tratamento 2: Utilizando processo PARFAIT	
Sistema Legado de Biblioteca	Sistema Legado de Oficina Eletrônica	Sistema Legado de Biblioteca	Sistema Legado de Oficina Eletrônica
Ex. Grupo 1/Participante 1	Ex. Grupo 2/Participante 2	Ex. Grupo 2/Participante 2	Ex. Grupo 1/Participante 1

Primeiramente é feita a reengenharia utilizando o processo "ad hoc" com todos os participantes (ou grupos) - Etapa 1. Em seguida, são fornecidos treinamentos do PARFAIT e das técnicas envolvidas para todos os participantes. Finalmente, realiza-se a reengenharia utilizando o processo PARFAIT com troca dos sistemas entre os participantes (ou grupos) - Etapa 2. A condução do experimento deve seguir essa seqüência para que não haja influência do treinamento na execução da reengenharia "ad hoc".

f) Instrumentação: Para a condução do experimento é necessário que os participantes possuam conhecimento em algumas técnicas. Para permitir isso, o pacote fornece toda a documentação necessária para que treinamentos sejam ministrados. Além disso, o

pacote disponibiliza alguns documentos e materiais que podem servir de apoio para a condução do experimento, conforme apresentados a seguir:

- Treinamento necessário: UML (diagrama de classes e diagrama de casos de uso), linguagem de padrões GRN, framework GREN, teste de software (critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite), diretrizes de reúso da abordagem ARTe, aplicação do Processo PARFAIT (exemplo de aplicação em uma função de um sistema legado de clínica médica).
- Documentos e materiais disponíveis para a condução do experimento: documento do processo PARFAIT (Cagnin et al., 2004g), da linguagem de padrões GRN (Braga et al., 1999), da abordagem ARTe (Capítulo 5), manual de uso da ferramenta GREN-Wizard (Braga, 2002a), manual de uso da ferramenta *GREN-WizardVersionControl* (Cagnin et al., 2004a), versão executável dos sistemas legados de controle de oficina eletrônica e de controle de biblioteca, documentação dos recursos de teste dos padrões da linguagem de padrões GRN.
- Formulários que devem ser preenchidos: formulário de perfil (Apêndice B), formulário de consentimento (Apêndice C), formulário de coleta de dados da Etapa 1 (Apêndice D) e da Etapa 2 (Apêndice E).

f) Avaliação da validade: Durante a avaliação dos resultados do experimento, é necessário considerar as seguintes ameaças de validade:

- **Validade de conclusão:** Alguns dados coletados são subjetivos e dependem da qualificação do engenheiro de software que está utilizando o processo; como os participantes são voluntários, podem abandonar o experimento em qualquer momento durante sua execução, podendo invalidar estatisticamente o experimento.
- **Validade interna:** o experimento será realizado pelos participantes sem restrição de dia, horário e local, portanto poderão aplicá-lo no momento que acharem mais adequado.
- **Validade de construção:** como o sucesso da aplicação do PARFAIT depende do treinamento fornecido, a qualidade do treinamento é imprescindível para que se obtenha dados significativos após a operação do experimento e para que seja possível rejeitar as hipóteses nulas.
- **Validade externa:** esse experimento pode ser conduzido no meio industrial ou acadêmico, com sistemas legados reais.

A.0.3 Fase de Operação

Após definir e planejar o experimento do pacote, foi dado início à preparação do material necessário para realizar o treinamento dos participantes envolvidos no estudo de caso. Nesse momento, foram confeccionadas as apresentações do treinamento. A maior parte do material de apoio, como relatórios e artigos técnicos, e manuais de ferramentas já existiam e foram disponibilizados, juntamente com as apresentações confeccionadas, em uma página da Internet [1] criada especificamente para divulgar o estudo de caso e a instrumentação necessária.

Posteriormente, o objetivo e a definição do pacote foram divulgados aos alunos de graduação e pós-graduação do ICMC-USP para a seleção dos participantes do estudo de caso. Os alunos que mostraram interesse foram contactados e todos participaram de uma reunião, cuja pauta foi explanar a importância da reengenharia de software e dirimir eventuais dúvidas sobre os objetivos e a condução do estudo de caso. Nessa reunião, elaborou-se também um cronograma de execução das atividades do estudo de caso juntamente com os participantes, para estipular a data de entrega das atividades de forma mais adequada e que não atrapalhasse os compromissos pessoais e acadêmicos de cada voluntário.

No final da reunião, os voluntários assinaram um termo de consentimento de participação no estudo de caso de caso e responderam um questionário de perfil. Esse questionário foi utilizado para a seleção dos componentes de cada grupo, seguindo a instrução apresentada no item **projeto do experimento** da fase de planejamento. Os questionários foram analisados e os grupos foram formados de acordo com a experiência dos voluntários em diversas áreas (veja Apêndice B). Assim, dois grupos, contendo dois componentes cada, foram formados com o mesmo nível de conhecimento para que não houvesse influência nos resultados finais.

Na Etapa 1 do estudo de caso (reengenharia "ad hoc"), os grupos tiveram que implementar o sistema alvo baseando-se em cinco sistemas exemplos gerados a partir da instanciação do framework GREN (sistemas de controle de locadora de carros, de loja de festas, de oficina mecânica, de vídeo locadora e de clínica veterinária). Além disso, exigiu-se documentação mínima do sistema legado para ser entregue, composta de: documento de requisitos, diagrama de casos de uso, diagrama de classes e documentação dos casos de teste utilizados para testar o sistema alvo.

Antes da condução da primeira etapa, foi dado treinamento, totalizando dez horas, nas seguintes técnicas: linguagem de programação Smalltalk e banco de dados MySQL (ambos utilizados na implementação do framework GREN); ferramenta VersionWeb (Soares et al., 2000), para controlar as versões dos artefatos produzidos; e teste de software, enfocando principalmente critérios de teste funcional. Não foi dado treinamento em UML, pois os participantes já tinham conhecimento prévio nessa técnica.

[1] URL: `http://www.labes.icmc.usp.br/~istela/experimentos/`

O código fonte dos dois sistemas legados foi disponibilizado na página do pacote de experimentação, indicando as funções de cada sistema que deveriam ser consideradas na reengenharia. Apenas algumas funções principais dos sistemas legados foram selecionadas devido a restrições de tempo para a condução do estudo de caso.

Depois do término da Etapa 1, os grupos entregaram os artefatos produzidos e deu-se início a Etapa 2. Nessa segunda etapa, um componente de um dos grupos desistiu do estudo de caso devido a compromissos acadêmicos. Mesmo assim, optou-se por continuar o estudo de caso. Na Etapa 2, como descrito na fase de planejamento (Subseção A.0.2), houve troca do sistema legado entre os grupos e o processo PARFAIT foi utilizado para apoiar a condução da reengenharia. Estipulou-se duas iterações para executar a reengenharia, priorizando os requisitos do legado que cada uma deveria conter. Assim, foi dado treinamento de algumas técnicas adicionais à primeira etapa, totalizado cinco horas: linguagem de padrões GRN (Braga et al., 1999), ferramenta de instanciação GREN-Wizard (Braga e Masiero, 2003), ferramenta de controle de versão *GREN-WizardVersionControl* (Capítulo 6), diretrizes de reúso da abordagem ARTe (Capítulo 5) e processo ágil de reengenharia PARFAIT (Capítulo 3).

Depois do término da Etapa 2, os dados coletados em ambas as etapas foram analisados. Na Etapa 1, nenhum grupo entregou a documentação dos casos de teste e muitas funcionalidades do sistema tomado como exemplo permaneceram no sistema alvo, descaracterizando o sistema alvo em relação ao sistema legado. Na Etapa 2, os grupos não entregaram todos os artefatos que deveriam ser criados com a aplicação do PARFAIT e também não utilizaram os recursos de teste agregados aos padrões da linguagem de padrões GRN e, conseqüentemente, não aplicaram as diretrizes para reúso de teste da abordagem ARTe. Um dos grupos, que tinha apenas um componente, não criou caso de teste e o outro grupo criou apenas doze casos de teste, sem o apoio de critérios. Por isso, tornou-se inviável realizar a fase de análise e de interpretação dos dados, já que os grupos não seguiram sistematicamente o processo PARFAIT na segunda etapa. No entanto, com a operação do estudo de caso foi possível notar que há necessidade de fornecer mais horas de treinamento em Smalltalk e motivar ainda mais a importância de reúso dos recursos de teste associados aos padrões da GRN. Além disso, há necessidade de acompanhamento da execução das atividades do processo PARFAIT para que os participantes não apliquem as atividades de forma "ad hoc" e sigam as atividades descritas de forma sistemática. Para isso, sugere-se a criação de uma ferramenta de apoio à aplicação do processo, que monitore a execução de cada uma de suas atividades. Observou-se também outra validade de conclusão que deve ser considerada, relacionada ao desinteresse e desmotivação dos participantes do experimento. Isso prejudica a execução do experimento, invalidando os dados coletados.

Apêndice

B

Formulário para Levantamento de Dados do Engenheiro de Software

Este formulário deve ser respondido por todos os participantes do experimento de reengenharia.

Identificação do engenheiro de software: _____

Data: ___/___/___

Este formulário pergunta a você algumas questões sobre o seu conhecimento e experiência.

Dados Pessoais

1. Última titulação:

Graduação	Especialização	Mestrado	Doutorado

Ano de conclusão: _____

Nome do Curso:_____

Caso ainda não tenha terminado a graduação, especifique:

- O semestre que está cursando: _____do total de _____semestres.
- Horário: Noturno ( ) Diurno ( )

- Quais disciplinas você já cursou na área de análise de sistemas/engenharia de software:

  _____

  _____

  _____

2. Qual sua maior área de interesse:

Engenharia de Software	Redes/Sistemas Distribuídos	Banco de Dados	Inteligência Artificial

Caso tenha outra especialidade, especificar? _____

Conhecimento Geral

3. Qual é sua experiência prévia com desenvolvimento de software orientado a objetos na prática? (Verifique o item que mais se aplica)

Eu nunca desenvolvi software	Eu desenvolvi software para mim mesmo	Eu desenvolvi software como parte de uma equipe, em um curso	Eu desenvolvi software como parte de uma equipe, na indústria

Quantos meses/anos de experiência você tem nesta prática? _____

Conhecimento Específico

4. Qual é sua experiência com padrões de software?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

5. Qual é sua experiência com a linguagem de padrões GRN?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

6. Qual é sua experiência com a linguagem de programação Smalltalk?

   Quantos meses/anos de experiência você tem nesta prática? _____

   Se você não tiver nenhuma experiência nessa linguagem especificar qual linguagem orientada a objetos você conhece: _____, e responda: Quantos meses/anos de

Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

   experiência você tem nesta prática? _____

7. Qual é sua experiência com frameworks orientados a objetos?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

8. Qual é sua experiência com o framework GREN?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

9. Qual é sua experiência com a ferramenta GREN-Wizard?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

Experiência com Reengenharia

10. Qual é sua experiência com reengenharia de sistemas?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

Experiência em Projeto de Sistemas

11. Qual é sua experiência em projetos de sistemas?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

( ) Análise Estruturada: _____meses/anos.

( ) Orientada a objetos: _____meses/anos.

( ) Outra:_____

12. Qual é sua experiência com Unified Modeling Language (UML)? Quantos meses/anos

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

de experiência você tem com UML? _____

Experiência em Teste

13. Qual é sua experiência em teste de software?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

14. Qual é sua experiência com a Técnica Particionamento em Equivalência?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

15. Qual é sua experiência com a Técnica Análise do Valor Limite?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

Outras Experiências

16. Qual é sua experiência com Gerenciamento de Configuração de Software? Quantos

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

meses/anos de experiência você tem? _____

17. Qual é sua experiência com Banco de Dados Relacional (SGBD)? Quantos meses/anos

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

de experiência você tem? _____

Especifique as matérias de banco de dados que você cursou: _____

Especifique os SGBDt's que você conhece: _____

18. Qual é sua experiência com o SGBD MySQL?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

Quantos meses/anos de experiência você tem com MySQL? _____

Experiência no Domínio

As respostas dessa seção serão utilizadas para entender a sua familiaridade com o domínio de Gestão de Recursos de Negócios.

19. Quanto você sabe sobre sistemas que controlam a manutenção de bens materiais, por exemplo, carro, eletrodomésticos, etc?

Não muito/não possui familiaridade	Algum/Familiar	Bastante/Muito familiar

20. Quanto você sabe sobre sistemas que controlam a comercialização de bens materiais e de consumo, por exemplo, compra e venda de produtos em uma loja de móveis ou em uma loja de produtos alimentícios?

Não muito/não possui familiaridade	Algum/Familiar	Bastante/Muito familiar

21. Quanto você sabe sobre sistemas que controlam a locação de bens materiais, por exemplo, locação de carro, locação de fitas de vídeo e DVD, etc?

Não muito/não possui familiaridade	Algum/Familiar	Bastante/Muito familiar

22. Qual é a sua experiência no desenvolvimento de sistemas no domínio de Gestão de Recursos de Negócios?

Nenhuma	Estudado em aula ou a partir de um livro	Praticado em um projeto de classe	Usado em um projeto ou na Indústria	Usado em vários projetos na indústria

Quantos meses/anos de experiência você tem nesta prática? _____

Sobre o Experimento

23. O que você espera obter participando do experimento?

Nada	Pouco	Razoável	Bastante

Observações ou comentários:

_____

_____

_____

Apêndice

C

Formulário de Consentimento

Título e propósito do projeto

O estudo de caso é chamado “Estudo de caso para avaliação do processo ágil de reengenharia PARFAIT” e tem como intuito avaliar PARFAIT quanto ao nível de reúso, qualidade do produto resultante e tempo de reengenharia em relação a processos de reengenharia “ad hoc”.

Declaração de idade

Eu declaro que tenho mais de 18 anos de idade e quero participar de um estudo de caso conduzido pela aluna de doutorado Maria Istela Cagnin, cujo orientador é o Prof. Dr. José Carlos Maldonado.

Procedimento

O estudo de caso será conduzido de forma independente, sendo que eu terei a liberdade de estipular meus próprios horários para a realização das atividades, desde que o controle dos horários de início e fim dessas atividades seja anotado em uma planilha fornecida. O estudo de caso se iniciará em __/__/____tendo término previsto para __/__/____.

Confidência

Toda informação coletada no estudo de caso é confidencial e meu nome não será identificado.

Benefícios e liberdade para desistir

Eu sei que não terei nenhum ganho pessoal participando do estudo de caso mas que o pesquisador espera saber mais sobre o processo PARFAIT como apoio para a reengenharia de sistemas procedimentais para o paradigma orientado a objetos. Eu sei que eu tenho a liberdade para perguntar qualquer questão ou para desistir da participação em qualquer hora sem penalidade e que eu terei acesso aos principais resultados do estudo de caso.

Responsáveis

Dr. José Carlos Maldonado Universidade de São Paulo ICMC-USP Av. Trabalhador São Carlense, 400 São Carlos, SP, 13560-970 Fone: (16) 3373-9669 email: jcmaldon@icmc.usp.br	Maria Istela Cagnin Universidade de São Paulo ICMC-USP Av. Trabalhador São Carlense, 400 São Carlos, SP, 13560-970 Fone: (16) 3373-9375 email: istela@icmc.usp.br

_____

Nome do participante

_____

Assinatura do participante Data: _/_/___

Apêndice

D

Formulário de Coleta de Dados Etapa 1

Instruções Gerais

- O projeto será feito EM GRUPO, não devendo haver nenhum tipo de comunicação a respeito do projeto entre os grupos.
- Dúvidas devem ser solucionadas diretamente com o responsável pelo experimento.
- O projeto de reengenharia, nesta primeira etapa do estudo, deve ser executado seguindo os conhecimentos dos alunos do grupo, e cada atividade deve ser anotada, bem como o tempo gasto.
- É necessário que o grupo entre em contato com o cliente para que o mesmo valide o produto que está sendo produzido.
- O questionário de perfil (Apêndice B) deve ser preenchido e entregue para o responsável do experimento.
- O questionário de coleta de dados da etapa 1 deve ser entregue no dia __/__/____.
- Na ficha F1 devem ser preenchidos os tempos gastos com quaisquer atividades relacionadas ao projeto, anotando separadamente o tempo gasto com cada tipo de atividade, por exemplo, familiaridade com as funcionalidades do sistema legado,

documentação do sistema, estudo da linguagem, implementação e teste. Ao término do projeto, faça uma estimativa do tempo total gasto com cada atividade (página 3 da ficha F1).

- A documentação do sistema deve conter:
  - Documento de requisitos,
  - Diagrama de casos de uso,
  - Diagrama de classes,
  - Documentação dos casos de teste (caso utilize algum critério de teste, especificar).
- O diagrama de classes deve ser feito à mão, anotando o tempo gasto para tal. Se for passar a limpo em uma ferramenta Case, anote esse tempo em separado.
- O tempo gasto para preencher as fichas não deve ser computado.
- Quaisquer dificuldades encontradas devem ser anotadas em uma folha, para facilitar responder o questionário Q1 no término do estudo de caso.

F1 - Coleta de Tempo da Reengenharia "Ad Hoc" - Pg. 1/4

Grupo:____

Data	**Início**	**Término**	**Tempo (pessoas/hora)**	**Atividades**

F1 - Coleta de Tempo da Reengenharia "Ad Hoc" - Pg. 2/4

Data	Início	Término	Tempo (pessoas/hora)	Comentários

F1 - Coleta de Tempo da Reengenharia "Ad Hoc" - Pg. 3/4

Resumo das atividades

Atividade	**Total de horas**

F1 - Coleta de Tempo da Reengenharia "Ad Hoc" - Pg. 4/4

Por favor, responda os seguintes tópicos:

Tópicos	**Total**
Quantidade total de funcionalidades do sistema legado submetidas a reengenharia	
Quantidade total de funcionalidades documentadas a partir de reúso	
Tempo gasto no entendimento e na documentação do sistema legado (i.e., tempo na elaboração do documento de requisitos, diagrama de casos de uso, diagrama de classes, documentação dos casos de teste).	
Quantidade total de classes e de métodos implementados na versão final do sistema	
Quantidade total de classes e de métodos reutilizados na versão final do sistema (a partir da implementação do sistema baseada em exemplos)	
Tempo gasto na implementação do sistema	
Quantidade total de casos de teste criados para validar o novo sistema	
Quantidade total de casos de teste reutilizados para validar o novo sistema	
Tempo gasto na fase de depuração e testes da nova aplicação	
Quantidade de tempo despendido para conduzir a reengenharia	

Tópicos que devem ser respondidos pelo cliente:

Tópicos	**Total**
Quantidade de erros encontrados pelos usuários durante a demonstração do novo sistema	
Quantidade de tarefas que o usuário deseja mas não consegue executar	
Quantidade de tarefas que o usuário consegue executar a partir do conjunto de tarefas determinadas	

Q1 - Questionário Individual para Participantes do Experimento

Nome:_____

A reengenharia foi bem sucedida, ou seja, atingiu as expectativas dos clientes e os objetivos iniciais? ( ) Sim ( ) Não

Justificativa: _____

_____

_____

Relate as dificuldades encontradas durante a aplicação da reengenharia "AD HOC".

_____

_____

_____

Apêndice

E

Formulário de Coleta de Dados Etapa 2

Instruções Gerais

- O projeto será feito EM GRUPO, não devendo haver nenhum tipo de comunicação a respeito do projeto entre os grupos.
- Dúvidas devem ser solucionadas diretamente com o responsável pelo experimento.
- O projeto de reengenharia, nesta segunda etapa do estudo, deve ser executado seguindo as atividades do processo ágil de reengenharia PARFAIT, e o tempo gasto em cada iteração de cada atividade e de cada marco de referência deve ser anotado na ficha F1. Ao término do projeto, faça uma estimativa do tempo total gasto com cada atividade e com cada marco de referência.
- É necessário que o grupo entre em contato com o cliente para que o mesmo valide o produto que está sendo produzido.
- O questionário de coleta de dados da etapa 2 deve ser entregue no dia __/__/____.
- O diagrama de classes deve ser feito à mão, anotando o tempo gasto para tal. Se for passar a limpo em uma ferramenta Case, anote esse tempo em separado.
- O tempo gasto para preencher as fichas não deve ser computado.

- Quaisquer dificuldades encontradas devem ser anotadas em uma folha, para facilitar responder o questionário Q1 no término do estudo de caso.

F1 - Coleta de Tempo da Reengenharia com PARFAIT - Pg. 1/4

Grupo:____

Data	**Início**	**Término**	**Tempo (pessoas/hora)**	**Atividades/Marcos de Referência**

F1 - Coleta de Tempo da Reengenharia com PARFAIT - Pg. 2/4

Data	Início	Término	Tempo (pessoas/hora)	Comentários

F1 - Coleta de Tempo da Reengenharia com PARFAIT - Pg. 3/4

Resumo da Seqüência de Iteração das Atividades

Nr. da sequência	**Atividade**	**Total de horas**	**Total de horas para inspeção**	**Qtde erros encontra-dos na inspeção**
	Total Geral:			

F1 - Coleta de Tempo da Reengenharia com PARFAIT - Pg. 4/4

Por favor, responda os seguintes tópicos:

Tópicos	**Total**
Quantidade total de funcionalidades do sistema legado submetidas a reengenharia	
Quantidade total de funcionalidades documentadas a partir de reúso	
Tempo gasto no entendimento e na documentação do sistema legado (i.e., atividades da fase de elaboração).	
Quantidade total de classes e de métodos implementados na versão final do sistema	
Quantidade total de classes e de métodos reutilizados na versão final do sistema (a partir da instanciação do framework GREN)	
Tempo gasto na implementação do sistema	
Quantidade total de casos de teste criados para validar o novo sistema	
Quantidade total de casos de teste reutilizados para validar o novo sistema	
Tempo gasto na fase de depuração e testes da nova aplicação	
Quantidade de tempo despendido para conduzir a reengenharia	

Tópicos que devem ser respondidos pelo cliente:

Tópicos	**Total**
Quantidade de erros encontrados pelos usuários durante a demonstração do novo sistema	
Quantidade de tarefas que o usuário deseja mas não consegue executar	
Quantidade de tarefas que o usuário consegue executar a partir do conjunto de tarefas determinadas	

Q1 - Questionário Individual para Participantes do Experimento

Nome:_____

A reengenharia foi bem sucedida, ou seja, atingiu as expectativas dos clientes e os objetivos iniciais? ( ) Sim ( ) Não

Justificativa: _____

_____

_____

Relate as dificuldades encontradas durante a aplicação da reengenharia com o apoio do PARFAIT.

_____

_____

_____