

PAULO CESAR ALLEBRANDT

*Reengenharia de Sistemas com RUP*  
*Estudo de Caso: APUFSC*

Florianópolis

2004

PAULO CESAR ALLEBRANDT

*Reengenharia de Sistemas com RUP*  
*Estudo de Caso: APUFSC*

Trabalho apresentado como requisito para  
obtenção do título em Bacharel em Ciências  
da Computação.

Orientador:  
Raul Sidnei Wazlawick

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis

2004

Trabalho de conclusão de curso sob o título de "*Reengenharia de Sistemas com RUP - Estudo de Caso: APUFSC*", defendido por Paulo Cesar Allebrandt e aprovada em 16 de Novembro de 2004, em Florianópolis, Estado de Santa Catarina, pela banca examinadora constituída pelos professores:

---

Prof. Dr. Raul Sidnei Wazlawick  
Orientador

---

Prof. Dr. Frank Siqueira

---

Prof. Dr. Ricardo Pereira e Silva

# *Agradecimentos*

Agradeço ao professor Raul Sidnei Wazlawick, meu orientador, pelo apoio e paciência em auxiliar no aprendizado das disciplinas necessárias para a realização deste trabalho.

Agradeço aos professores Frank Siqueira e Ricardo Pereira e Silva por terem aceitado fazer parte da banca do trabalho.

Agradeço aos professores do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina (INE-UFSC), pela base de conhecimento proporcionada nos últimos quatro anos, que possibilitou a realização deste trabalho.

Agradeço à turma CCO/002 pela amizade demonstrada durante todo o período em que estivemos juntos seja em festas ou em horas de estudo na biblioteca.

Agradeço aos meus sócios na AgroDigital, Daniel e Leonardo, pela compreensão nas minhas faltas justificadas por este trabalho, por terem lido e criticado o trabalho durante seu desenvolvimento e por terem aceitado ter algumas conversas que foram bastante esclarecedoras em alguns pontos do trabalho.

Agradeço à minha família, meus pais Paulo e Teresinha, e meus irmãos Ricardo e Camila, por todo o apoio emocional, financeiro e de qualquer outra natureza. Se não fosse por seu suporte tudo teria sido muito mais difícil.

Agradeço em especial à minha noiva, Jamile, por ter estado presente em todos os momentos, por ter tido paciência e compreensão nas horas de correria para deixar este trabalho pronto e por todo o amor e carinho que sempre demonstrou, fazendo com que tudo ficasse mais fácil.

# *Resumo*

O presente trabalho é uma experiência no uso da metodologia de desenvolvimento Rational Unified Process® (RUP) em um processo de reengenharia de sistemas. O RUP é considerado uma metodologia altamente adaptável, o que é comprovado através dos ajustes necessários no processo normal de RUP, para se realizar a reengenharia.

O software alvo do processo de reengenharia foi o sistema da Associação de Professores da UFSC (APUFSC), originalmente desenvolvido em Microsoft Access® com pouquíssima documentação, o que dificulta sua manutenção. A motivação para esta reengenharia foi a transição para uma plataforma não-proprietária e a geração de documentação do sistema, facilitando futuros esforços em manutenção.

# *Abstract*

This paper presents an experience in the use of Rational Unified Process (RUP) development methodology in a system reengineering process. The RUP is considered a highly adaptable methodology, what is verified through the necessary adjustments in the RUP's normal process for accomplishing the reengineering.

The target software of the reengineering process was the UFSC's Professors Association (APUFSC) system, originally developed in Microsoft Access® with little documentation, which makes maintenance hard. The motivation to make this system's reengineering was the transition to a non-proprietary platform and the generation of system's documentation making it easier future maintenance efforts.

# *Lista de Figuras*

• Figura 1: Modelo BPR. ....	21
• Figura 2: Modelo de processo de reengenharia de software. ....	26
• Figura 3: Estrutura geral do RUP em duas dimensões. ....	36
• Figura 4: Modelo conceitual com informações sobre a relação entre professores, departamentos e centros. ....	40
• Figura 5: Tela inicial do atual sistema da APUFSC. ....	61
• Figura 6: Interface principal do caso de uso Lançar Receitas e Despesas. ....	72
• Figura 7: Interface de suporte do caso de uso Lançar Receitas e Despesas. ....	72
• Figura 8: Diagrama de sequência do caso de uso Lançar Receitas e Despesas. ...	73
• Figura 9: Primeira parte do modelo conceitual do sistema. ....	74
• Figura 10: Segunda parte do modelo conceitual do sistema. ....	75
• Figura 11: Diagrama de colaboração referente ao contrato carregaBancos. ....	78
• Figura 12: Diagrama de colaboração referente ao contrato carregaTiposDeLançamentos. ....	78
• Figura 13: Diagrama de colaboração referente ao contrato identificaBanco. ....	78
• Figura 14: Diagrama de colaboração referente ao contrato carregaDataInicial. ...	79
• Figura 15: Diagrama de colaboração referente ao contrato carregaAgencias. ...	79
• Figura 16: Diagrama de colaboração referente ao contrato identificaAgencia. ...	79
• Figura 17: Diagrama de colaboração referente ao contrato carregaContas. ....	79
• Figura 18: Diagrama de colaboração referente ao contrato identificaConta. ....	80
• Figura 19: Diagrama de colaboração referente ao contrato carregaLancamentos. ...	80

- Figura 20: Diagrama de colaboração referente ao contrato enviaLancamento. . . 80
- Figura 21: Diagrama de classes de projeto com as classes referentes ao caso de uso Lançar Receitas e Despesas. .... 81



# *Lista de Tabelas*

- Tabela 1: Exemplo de tabela de conceitos. .... 40
- Tabela 2: Listagem de algumas tabelas do sistema antigo da APUFSC. .... 58
- Tabela 3: Requisitos suplementares para o processo de reengenharia. .... 67
- Tabela 4: Listagem de casos de uso identificados no levantamento de requisitos. 68
- Tabela 5: Listagem dos conceitos identificados no levantamento de requisitos. .. 69

# *Lista de Siglas*

- **APUFSC** - Associação de Professores da UFSC
- **BPR** - Business Process Reengineering (Processo de Reengenharia de Negócios)
- **RUP** - Rational Unified Process® (Processo Unificado Rational)
- **SEI** - Software Engineering Institute (Instituto de Engenharia de Software)
- **TI** - Tecnologia da Informação
- **UML** - Unified Modeling Language (Linguagem de Modelagem Unificada)
- **UP** - Unified Process (Processo Unificado)

# *Sumário*

<b>1</b>	<b>Introdução</b>	p. 14
1.1	Objetivos . . . . .	p. 16
1.1.1	Gerais . . . . .	p. 16
1.1.2	Específicos . . . . .	p. 16
1.2	Justificativas . . . . .	p. 16
1.3	Ferramentas Utilizadas . . . . .	p. 17
1.4	Estrutura do Trabalho . . . . .	p. 17
<b>2</b>	<b>Reengenharia de Sistemas</b>	p. 18
2.1	Business Process Reengineering . . . . .	p. 19
2.1.1	Processos de Negócio . . . . .	p. 19
2.1.2	Princípios da BPR . . . . .	p. 20
2.1.3	Um modelo de BPR . . . . .	p. 21
2.1.4	Fatores de sucesso para BPR . . . . .	p. 22
2.1.5	Um caso de sucesso no uso de BPR . . . . .	p. 24
2.2	Reengenharia de Software . . . . .	p. 25
2.2.1	Modelos de processo de reengenharia de software . . . . .	p. 26
2.2.2	Razões que levam ao fracasso de uma reengenharia . . . . .	p. 29
2.2.3	Engenharia Reversa . . . . .	p. 31
<b>3</b>	<b>Rational Unified Process (RUP)</b>	p. 34
3.1	Principais fluxos de trabalho do processo . . . . .	p. 36

3.1.1	Modelagem do Negócio . . . . .	p. 37
3.1.2	Levantamento de Requisitos . . . . .	p. 37
3.1.2.1	Requisitos . . . . .	p. 38
3.1.2.2	Organização dos Requisitos . . . . .	p. 39
3.1.3	Análise . . . . .	p. 41
3.1.3.1	Expansão dos Casos de Uso . . . . .	p. 41
3.1.3.2	Construção do Modelo Conceitual . . . . .	p. 42
3.1.3.3	Elaboração dos contratos . . . . .	p. 43
3.1.4	Projeto . . . . .	p. 43
3.1.4.1	Diagramas de Interação . . . . .	p. 44
3.1.4.2	Diagramas de Classes . . . . .	p. 45
3.1.5	Implementação . . . . .	p. 45
3.1.6	Testes . . . . .	p. 46
3.1.7	Implantação . . . . .	p. 46
3.2	Principais fluxos de trabalho de suporte . . . . .	p. 47
3.2.1	Gerenciamento de configuração e mudança . . . . .	p. 47
3.2.2	Gerenciamento do projeto . . . . .	p. 47
3.2.3	Ambiente . . . . .	p. 48
3.3	Fases . . . . .	p. 48
3.3.1	Concepção . . . . .	p. 48
3.3.2	Elaboração . . . . .	p. 49
3.3.3	Construção . . . . .	p. 51
3.3.4	Transição . . . . .	p. 52
3.4	Iterações . . . . .	p. 53
3.5	Reengenharia com RUP . . . . .	p. 54
3.5.1	Fluxos de Trabalho . . . . .	p. 54

3.5.1.1	Levantamento de Requisitos . . . . .	p. 54
3.5.1.2	Análise . . . . .	p. 55
3.5.1.3	Projeto . . . . .	p. 55
3.5.1.4	Implantação . . . . .	p. 56
3.5.2	Fases . . . . .	p. 56
<b>4</b>	<b>Estudo de Caso: APUFSC</b>	<b>p. 57</b>
4.1	Reengenharia do Sistema da APUFSC . . . . .	p. 57
4.1.1	Estrutura do atual sistema . . . . .	p. 58
4.1.2	Levantamento de Requisitos . . . . .	p. 60
4.1.3	Análise . . . . .	p. 62
4.1.4	Projeto . . . . .	p. 63
4.1.5	Implementação . . . . .	p. 64
<b>5</b>	<b>Conclusão</b>	<b>p. 65</b>
	<b>Apêndice A – Documentação</b>	<b>p. 67</b>
A.1	Levantamento de Requisitos . . . . .	p. 67
A.1.1	Requisitos suplementares . . . . .	p. 67
A.1.2	Listagem de casos de uso . . . . .	p. 67
A.1.3	Listagem dos conceitos . . . . .	p. 68
A.1.4	Listagem das consultas . . . . .	p. 69
A.2	Casos de Uso Expandidos . . . . .	p. 71
A.3	Modelo Conceitual . . . . .	p. 74
A.4	Contratos . . . . .	p. 76
A.5	Diagramas de Colaboração . . . . .	p. 78
A.6	Diagrama de Classes de Projeto . . . . .	p. 81
A.7	Implementação da Camada de Domínio . . . . .	p. 81

A.7.1	Classe TiposDeLancamentos . . . . .	p. 81
A.7.2	Classe Lancamento . . . . .	p. 82
A.7.3	Classe ContaBancaria . . . . .	p. 83
A.7.4	Classe Agencia . . . . .	p. 84
A.7.5	Classe Banco . . . . .	p. 86
A.7.6	Classe APUFSC . . . . .	p. 87

<b>Referências</b>		p. 90
--------------------	--	-------

# 1 *Introdução*

Com o surgimento da informática diversas empresas tiveram de rever seus processos a fim de se adaptar às facilidades introduzidas pelos computadores e pelos sistemas computacionais, processo este que se estende, segundo Bartié [BAR 02, p. 4], até os dias de hoje. Porém, o início da atividade de desenvolvimento de softwares foi marcado pelo que se pode chamar de “artesanato” em software. “Em geral, os cronogramas de produção de software atrasavam, os custos excediam enormemente os orçamentos e os produtos finais não eram confiáveis” [DEI 01, p. 66]. Isto porque os sistemas eram desenvolvidos sem uma metodologia bem definida, gerando pouca ou nenhuma documentação sobre o desenvolvimento, o que tornava a manutenção dos sistemas extremamente complexa e dispendiosa.

“Mesmo nos produtos de software desenvolvidos com os melhores processos, alguns defeitos escapam até chegarem à versão de operação, para serem então descobertos pelos usuários. [...] A rigor, a tarefa de manutenção do software consiste na remoção dos defeitos remanescentes após o fim do projeto de desenvolvimento. Esse tipo de manutenção é chamado de **manutenção corretiva**. Ela não inclui somente a correção de problemas no código, mas as modificações conseqüentes em todos os artefatos correlatos” [dPPF 03, p. 298].

“Na prática, outros problemas acabam sendo tratados como manutenção; como por exemplos, modificações nas interfaces de usuário, pequenas expansões funcionais e alterações para melhoria do desempenho. Essas modificações podem ser feitas para adaptar o produto a variações nos processos de negócio (chamada de **manutenção adaptativa**), ou para introduzir melhorias solicitadas pelos usuários (**manutenção perfectiva**). Em ambos os casos, existe alteração de requisitos” [dPPF 03, p. 298].

Quando o número de novos requisitos, ou alterações em requisitos, cresce muito, ou quando as tarefas auxiliadas pelo sistema mudam tornando-o pouco eficiente ou simplesmente inutilizável, surge a necessidade de uma nova solução, um novo sistema. Esta

renovação do sistema pode ser feita via uma reengenharia, com a qual “você criará um produto com funcionalidades adicionais, melhor performance e confiabilidade, e facilidade de manutenção melhorada”<sup>1</sup>

Como pode ser visto em Pressman [PRE 00, p. 804], a reengenharia é bastante útil para a migração de sistemas antigos feitos com pouca ou nenhuma documentação, em uma época em que as principais preocupações na construção de um software eram com o tamanho do código e com os sistemas de armazenamento. Também é possível lançar mão da reengenharia para a migração de sistemas locais para sistemas baseados na Internet, como proposto por Alcântara em [ALC 02]. Tudo isso mostra a importância do estudo da reengenharia, bem como de metodologias para torná-la mais eficiente.

“O Rational Unified Process® é um processo de engenharia de software. Ele provê uma abordagem disciplinada para determinação de tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de software de alta qualidade que concilie-se com as necessidades do usuário final, dentro de prazo e orçamento previsíveis”<sup>2</sup>.

“Algumas pessoas afirmam que o Rational Unified Process® (RUP) é útil apenas [...] para o desenvolvimento de um sistema totalmente novo [...]. Eles afirmam que não pode ser usado para promover o desenvolvimento ou evolução de um sistema legado”<sup>3</sup>. Porém, muitos artefatos de RUP podem ser usados para o processo de reengenharia partindo-se de uma engenharia reversa. Neste caso, engenharia reversa significa algo como “tentar identificar, extrair, ou recriar informação suficiente para possibilitá-lo a proceder quase como se o projeto fosse desenvolvido originalmente usando-se RUP”<sup>4</sup>.

---

<sup>1</sup>“you’ll create a product with added functionality, better performance and reliability, and improved maintainability” [PRE 00, p. 779].

<sup>2</sup>“The Rational Unified Process® is a Software Engineering Process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget” [RAT 98, p. 1].

<sup>3</sup>“Some people claim that the Rational Unified Process® (RUP) is only useful for [...] the development of a brand new system [...]. They contend that it cannot be used for further development or evolution of a ”legacy” system.” [KRU 03].

<sup>4</sup>“trying to identify, extract, or recreate enough information to enable you to proceed almost as if the project had been originally developed using the RUP” [KRU 03].



## 1.1 Objetivos

### 1.1.1 Gerais

Este trabalho tem como objetivo documentar uma experiência no uso do Rational Unified Process® em um processo de reengenharia de sistemas através de um estudo de caso realizado com o sistema da APUFSC.

### 1.1.2 Específicos

- Avaliar a adaptabilidade da metodologia RUP utilizando-o como ferramenta de reengenharia.
- Realizar a reengenharia do sistema da APUFSC.
- Documentar as especificidades de uma reengenharia a partir de um sistema desenvolvido em Microsoft Access®.

## 1.2 Justificativas

A importância da reengenharia já foi discutida, porém vale salientar que são poucos os estudos na aplicação das metodologias de engenharia de software no processo de reengenharia de software. É verdade que os processos se assemelham em muita coisa, já que os artefatos usados são os mesmos e a reengenharia difere, basicamente, na fonte de suas informações, onde conta com um sistema legado, porém alguns passos podem ser suprimidos ou modificados quando se trata de uma reengenharia, de forma que é possível se fazer uma reavaliação das metodologias tornando-as mais adaptadas à tarefa de reengenharia.

O sistema da APUFSC foi escolhido por ser um sistema feito sem um paradigma bem definido, por desenvolvedores com pouca experiência na ferramenta de desenvolvimento (MS Access®) e com ferramentas precárias de engenharia de software, ou seja, é um sistema com recursos precários para sua manutenção. Outro ponto que motiva a reengenharia do sistema da APUFSC é o fato de ele estar vinculado a uma ferramenta proprietária, o Microsoft Access®, o que deseja-se mudar com a adoção de alguma tecnologia livre, como Java, por exemplo.

A metodologia escolhida para este trabalho foi o RUP por ser bastante difundida e usada, contando com vasta literatura. Além disso, RUP é um processo altamente

adaptável e voltado à garantia de qualidade do produto final, o que facilita a adaptação para um bom processo de reengenharia.

### **1.3 Ferramentas Utilizadas**

Toda a documentação gerada durante o processo de reengenharia foi feita utilizando-se um editor de textos e, para os diagramas, o Jude Take, uma ferramenta grátis desenvolvida pelo grupo japonês ObjectClub (<http://www.objectclub.jp>).

### **1.4 Estrutura do Trabalho**

O capítulo 2 conceitua Reengenharia de Sistemas partindo do conceito de BPR (Business Process Reengineering) e chegando até os conceitos de Engenharia Reversa e Remanufatura.

O capítulo 3 faz um aprofundamento no estudo de RUP, mostrando suas fases, seus fluxos e atividades, e termina fazendo uma análise do processo visando a reengenharia.

No capítulo 4 é relatado o processo de reengenharia do sistema da APUFSC, partindo-se de uma análise da estrutura do sistema atual da APUFSC e terminando com a descrição dos passos realizados nos fluxos de levantamento de requisitos, análise e projeto.

O apêndice mostra a documentação gerada durante o processo de reengenharia do sistema da APUFSC.

## 2 *Reengenharia de Sistemas*

Quando um produto físico quebra, fica velho ou, por qualquer outra razão já não serve mais adequadamente aos seus propósitos e seu conserto se torna dispendioso demais, a atitude mais natural é se trocar o produto por um similar mais novo. No caso de um software a situação não se resolve tão facilmente. Quando um software começa a demonstrar sinais de que já não atende mais aos requisitos de uma organização, ou seja, quando os requisitos da organização mudam muito em relação ao que foi concebido para o software, surge a necessidade de se construir um novo sistema similar ou adaptar o sistema existente, adicionando funcionalidades, melhorando a performance e melhorando a manutenção. Isso é o que pode ser chamado de reengenharia.

A idéia de reengenharia surgiu, segundo Pressman [PRE 00, p. 799], em um artigo de Michael Hammer para o Harvard Business Review, onde foi lançada uma revolução na forma de pensar sobre processos de negócios e computação. Nesse artigo, Hammer chama a atenção ao fato de que não se deve simplesmente “encaixar processos antigos em silício e software”<sup>1</sup>, mas deve-se fazer um trabalho mais profundo, “nós deveríamos fazer a ‘reengenharia’ do nosso negócio: usar o poder da tecnologia da informação moderna para redesenhar radicalmente nossos processos de negócio para alcançar dramáticos aprimoramentos em sua performance”<sup>2</sup>.

O que pode-se notar então, segundo Pressman [PRE 00, p. 820], é que a reengenharia pode ocorrer em dois diferentes níveis de abstração. A nível de negócios, onde a reengenharia é focada nos processos de negócio com o intento de alterá-los aumentando a competitividade da organização em determinada área, a chamada *Business Process Reengineering* (BPR). E a nível de software, onde a reengenharia atua analisando um sistema buscando reestruturá-lo ou reconstruí-lo com maior qualidade.

---

<sup>1</sup> “[...] embedding outdated processes in silicon and software” [HAM 90].

<sup>2</sup> “We should ‘reengineer’ our businesses: use the power of modern information technology to radically redesign our business processes in order to achieve dramatic improvements in their performance” [HAM 90].

## 2.1 Business Process Reengineering

Segundo Jacobson [JAC 94, p. ix], Business Process Reengineering foi definido por Hammer em [HAM 93] como “a reavaliação fundamental e um redesenho radical dos processos de negócio para alcançar dramáticas melhorias em medidas de performance contemporâneas críticas como custo, qualidade, serviço e velocidade”<sup>3</sup>.

“BPR implica que você tenha compreensão de toda a operação existente e avalie por que você faz o que faz, por que você faz o que faz da maneira como faz, por que... Resumindo, BPR requer que você questione toda a operação e tente redesenhá-la de forma a usar novas tecnologias para servir melhor ao seu cliente”<sup>4</sup>.

Em outras palavras, BPR significa “questionar todo o negócio existente - ou pelo menos os principais processos - e tentar encontrar meios completamente novos de reconstruí-los”<sup>5</sup>. Davenport [DAV 93], citado em [JAC 94, p. 14], chama isso de inovação de processo, “maior redução em custo ou tempo de processo, ou maior melhoria na qualidade, flexibilidade ou níveis de serviço ou outros objetivos de negócio”<sup>6</sup>

Segundo Pressman [PRE 00, p. 801], BPR deve ocorrer em todo o negócio, começando-se por identificar os grandes objetivos da organização e descendo para o detalhamento de processos de negócio, sub-processos de negócio e tarefas relacionadas a cada processo.

### 2.1.1 Processos de Negócio

O foco da BPR são os processos de negócio, “de forma simples, um processo de negócio é um conjunto de atividades internas executadas para servir a um cliente. O propósito de cada processo de negócio é oferecer a cada cliente o produto ou serviço certo, com alto grau de performance nas medidas de custo, longevidade, serviço e qualidade”<sup>7</sup>. Vale notar que o termo ‘cliente’ pode ser entendido de forma estendida, abrangendo não apenas um

<sup>3</sup> “the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed”.

<sup>4</sup> “BPR implies that you take a comprehensive view of the entire existing operation and think through why you do what you do, why you do what you do the way you do it, and why... In short, BPR requires that you question the entire existing operation and try to redesign it in a way that uses new technology to serve your customer better” [JAC 94, p. ix].

<sup>5</sup> “[...] question the entire existing business - or at least its most important processes - and try to find completely new ways of reconstructing them[...]” [JAC 94, p. 14].

<sup>6</sup> “major reductions in process cost or time, or major improvements in quality, flexibility, service levels or other business objectives”.

<sup>7</sup> “Put simply, a business process is the set of internal activities performed to serve a customer. The purpose of each business process is to offer each customer the right product or service (that is the right deliverable), with a high degree of performance measured against cost, longevity, service and quality. [JAC 94, p. 3]”

cliente normal, como também um outro processo externo à organização, como um parceiro ou servidor terceirizado.

### 2.1.2 Princípios da BPR

Hammer, em [HAM 90], sugeriu alguns princípios para guiar as atividades de BPR:

- *Organize resultados, não tarefas.* Muitas organizações possuem atividades distribuídas em pequenos grupos, assim ninguém tem responsabilidade individual sobre um resultado, porém também se torna difícil determinar o status do trabalho e encontrar a fonte de um problema no processo. BPR deve organizar processos que evitem esse problema.
- *Deixe que aqueles que usam a saída do processo executem o processo.* A intenção é deixar que aquele que precisa de um resultado possa controlar todas as variáveis necessárias para permitir alcançá-lo. Quanto menos separados as partes envolvidas num processo, mais suave o caminho para um rápido resultado.
- *Incorpore o processamento de uma informação ao trabalho daqueles que geram a informação.* Como a tecnologia da informação (TI) está cada vez mais distribuída, é possível fazer com que aqueles que geram a informação possam processá-la, evitando assim custos de comunicação e descentralizando os custos de processamento.
- *Trate recursos geograficamente dispersos como se estivessem centralizados.* Com as sofisticadas formas de comunicação via computadores pode-se trabalhar com um grupo disperso de profissionais como se eles estivessem no mesmo “escritório virtual”.
- *Ligue atividades paralelas ao invés de integrar seus resultados.* Quando diferentes grupos trabalham em paralelo é essencial criar um processo em que os grupos se mantenham em comunicação e coordenação, evitando assim problemas de coordenação.
- *Ponha os pontos de decisão onde o trabalho é desempenhado, e coloque o controle dentro dos processos.*
- *Capture um dado apenas uma vez.* Os dados devem ser armazenados on-line, de forma que só tenham que ser inseridos uma vez.

### 2.1.3 Um modelo de BPR

Pressman, em [PRE 00, p. 802], indica um modelo de BPR. Este modelo é iterativo e evolucionário, não possui necessariamente um começo e um fim.



Figura 1: Modelo BPR [PRE 00, p. 803]

A figura 1 mostra as seis atividades básicas e suas interligações:

**Definição do negócio.** Os objetivos do negócio são identificados dentro do contexto de quatro guias principais: redução de custos, redução de tempo, aumento de qualidade e desenvolvimento e melhoramento de pessoal.

**Identificação de processos.** São identificados os processos considerados críticos para se alcançar os objetivos definidos da “Definição do negócio”. Os processos devem ser ordenados de alguma forma, por importância, necessidade de mudança ou qualquer outra variável.

**Avaliação de processos.** Os processos existentes são analisados e medidos. As tarefas dos processos são identificadas e seus custos e tempo consumidos são avaliados e seus problemas de qualidade e/ou performance são isolados.

**Design e especificação de processos.** São preparados casos de uso<sup>8</sup> para cada processo que deve ser re-projetado. A partir dos casos de uso um novo grupo de tarefas é desenvolvido para cada processo.

<sup>8</sup>Veja as seções 3.1.2.2, página 39 e 3.1.3.1, página 41

**Prototipação.** Antes de integrar os novos modelos de processos ao negócio existente, tais processos devem ser testados para que refinamentos sejam feitos.

**Refinamento e instanciação.** Com base nos resultados dos testes da prototipação os processos devem ser refinados e então instanciados dentro do negócio.

#### 2.1.4 Fatores de sucesso para BPR

Segundo Jacobson<sup>9</sup>, “os riscos de fracasso num projeto de reengenharia são extraordinariamente grandes, Hammer estima que entre 50 e 70% dos projetos não alcançam o sucesso - ou seja, não alcançam melhorias dramáticas”<sup>10</sup>.

Para Pressman, “BPR pode funcionar, se aplicada por pessoas motivadas e treinadas que sabem que o processo de reengenharia é uma atividade continuada. Se BPR for conduzida efetivamente, os sistemas de informação são mais bem integrados no processo de negócio. A reengenharia de sistemas antigos pode ser avaliada dentro do contexto da estratégia de todo o negócio, e prioridades para a reengenharia de software podem ser estabelecidas de forma inteligente”<sup>11</sup>.

Kathleen Flynn, em [FLY 93], apresenta alguns fatores críticos para o sucesso de um projeto de reengenharia:

- *Motivação.* Os motivos para iniciar um projeto de reengenharia devem estar bem definidos. A alta administração deve estar absolutamente convencida de que os esforços em reengenharia levarão a consideráveis melhorias e devem entender que algumas estruturas serão despedaçadas por bons motivos. Para garantir o sucesso, a administração deve se comprometer completamente, a empresa deve ser totalmente envolvida e suas melhores cabeças devem estar empenhadas no time de reengenharia. Além disso o pessoal envolvido deve ter plena consciência das razões pelas quais o projeto está sendo desenvolvido, ou seja, devem compreender os problemas a serem atacados, também devem aceitar suas novas tarefas e serem treinadas para desempenhá-las.

---

<sup>9</sup> “the risk of failure a reengineering project are extraordinarily great. Hammer estimates that between 50 and 70% of the projects don’t succeed - that is, they do not achieve dramatic improvements” [JAC 94, p. 18].

<sup>10</sup> Aqui Jacobson fala apenas de BPR, não de reengenharia de software.

<sup>11</sup> “BPR can work, if it is applied by motivated, trained people who recognize that process reengineering is a continuous activity. If BPR is conducted effectively, information systems are better integrated into the business process. Reengineering older applications can be examined in the context of a broad-based business strategy, and priorities for software reengineering can be established intelligently” [PRE 00, p. 804].

- *Liderança.* Um condutor que seja parte da administração da companhia, que tenha autoridade e goze de confiança nas organizações envolvidas deve encabeçar o projeto e assumir a responsabilidade. Mesmo antes de começar o projeto, o líder deve ter consciência das dificuldades da empreitada: deve resistir às pressões da organização e deve convencer a todos de que o projeto não é apenas vital, mas essencial para a sobrevivência da organização.
- *Posse de toda a organização.* A organização envolvida, incluindo pessoal em todos os níveis, deve ser persuadida a assumir uma posse comum do trabalho que trará as alterações. Alguns gerentes intermediários podem trazer problemas, seja por resistirem a mudanças em processos definidos por eles, seja por estarem acomodados com os processos antigos, estes serão os mais difíceis de se convencer da necessidade e da importância das mudanças. Já o pessoal de “chão de fábrica” pode ser mais facilmente convencido.
- *Visão.* Os novos objetivos da companhia devem ser eloqüentes e facilmente compreendidos por todos os envolvidos.
- *Foco.* O trabalho de mudanças na companhia deve focar os objetivos de maior prioridade, e os recursos devem ser destinados a esses objetivos. (Nunca morda mais do que pode mastigar...)
- *Papéis e responsabilidades bem definidos.*
- *Produtos tangíveis.* Os resultados da reengenharia devem ser concretos e realizáveis, como objetivos e missões reformulados, novos fluxos de trabalho, um modelo de processo, um plano organizacional e um modelo de dados.
- *Suporte tecnológico.* Suporte na forma de métodos e ferramentas são indispensáveis para o trabalho de reengenharia. A engenharia de negócios geralmente envolve a construção de um sistema de informação para o novo negócio. Esta é uma área de risco, pois o sistema de informação deve ser bem construído para evitar futuros problemas<sup>12</sup>.
- *Aconselhamento de um especialista.* Uma consultoria pode auxiliar um time inexperiente em reengenharia, porém deve-se tomar cuidado pois o consultor não deve

---

<sup>12</sup>Aqui Flynn comenta o problema da falta de uso de metodologias para o desenvolvimento de sistemas. Na época, o Software Engineering Institute (SEI) estimava que 85% dos softwares eram desenvolvidos sem qualquer tipo de metodologia.



controlar, mas dar suporte ao time. O consultor não deve fazer com que o pessoal da administração fique passivo perante o processo de reengenharia.

- *Aceitar o risco.* Você não pode ganhar numa loteria sem sequer jogar.

Para Jacobson [JAC 94, p. 20], os riscos de um projeto de reengenharia podem ser diminuídos drasticamente se um processo formal for usado. Para Jacobson a orientação a objetos é o melhor caminho pois define uma linguagem padrão para a definição dos negócios e ainda auxilia na formalização do processo de reengenharia de negócios.

### 2.1.5 Um caso de sucesso no uso de BPR

Antunes [ANT ] apresenta alguns casos de sucesso no uso de BPR, entre eles o caso Mutual Benefit Life Insurance (MBL):

“A MBL foi uma das empresas que utilizou, com bastante sucesso, a reengenharia nos seus processos.

“Esta seguradora tinha um modo de funcionamento idêntico a todas as outras. Uma aplicação tinha cerca de 30 passos, passando por 5 departamentos e envolvendo 19 pessoas. Na melhor das hipóteses, a MBL processava uma aplicação em 24 horas, embora normalmente demora-se entre 5 a 24 dias.

“Um segurador estimou que embora uma aplicação demorasse 22 dias a ser processada, apenas era “trabalhada” 17 minutos, sendo a maior parte do tempo gasto na passagem de informação de departamento para departamento.

“O presidente da MBL perante este cenário resolve então incumbir uma equipe de tentar aumentar a performance da empresa em cerca de 60% na produtividade. A equipe, depois de ter olhado primeiramente para as tecnologias de informação como forma de atingir o objetivo, facilmente concluiu que, através de bases de dados partilhadas e redes de computadores poderiam tornar disponíveis, a uma única pessoa, uma grande variedade de informação. Além disso, sistemas periciais poderiam ajudar pessoas com pouca experiência a tomar decisões.

“A MBL decide então eliminar uma série de posições de trabalho e cria uma nova posição chamada “Case Manager”. Estes case managers têm total responsabilidade por uma aplicação, isto é, executam todas as tarefas associadas a uma aplicação, utilizando para isso poderosas Workstations correndo sistemas periciais e ligadas a uma Mainframe.

“Apenas em alguns casos o case manager recorre, por exemplo, à ajuda de um médico que apenas funcionará como um consultor ou aconselhador.

“Uma aplicação passou então a poder ser processada, na melhor das hipóteses, em apenas 4 horas, demorando normalmente entre 2 a 5 dias. Desta forma a MBL conseguiu um aumento estrondoso na sua produtividade”.

## 2.2 Reengenharia de Software

Um aplicação serve a um determinado negócio durante vários anos. Durante todo esse tempo erros foram sendo corrigidos e adaptações e melhoramentos foram feitos constantemente. Tal trabalho foi feito com a melhor das intenções, mas sempre sem cuidados com as boas práticas de engenharia de software. A aplicação agora se tornou instável, ainda pode ser corrigida, porém o trabalho demandado para se entender o código, achar o problema e fazer as devidas correções é muito grande e cada alteração pode trazer alguns efeitos colaterais indesejados.

Segundo Pressman [PRE 00, p. 804], este é um cenário bastante comum, e uma boa explicação para todo esse esforço em manutenção convergindo para resultados insatisfatórios pode ser encontrado em Osborne e Chikofsky<sup>13</sup>:

“Muitos dos softwares dos quais dependemos hoje têm de 10 a 15 anos de idade. Mesmo quando eles foram criados com as melhores técnicas de projeto e codificação conhecidas na época (e a maioria não foi), eles foram criados quando tamanho do programa e espaço de armazenamento eram as principais preocupações. Eles eram então migrados para novas plataformas, ajustados para mudanças de máquina e sistema operacional e melhorados para suprir as necessidades dos usuários - tudo sem atenção suficiente para a arquitetura geral.

“O resultado é uma estrutura de projeto pobre, codificação pobre, lógica pobre, e documentação pobre do sistema que agora deve-se manter rodando...”

Frente a um cenário como esse a reengenharia de software surge como uma proposta bastante interessante: “reconstruir o sistema, adicionando funcionalidades, melhorando a

---

<sup>13</sup> “Much of the software we depend on today is on average 10 to 15 years old. Even when these programs were created using the best design and coding techniques know at the time [and most were not], they were created when program size and storage space were principle concerns. They were then migrated to new plataforms, adjusted for changes in machine and operating system technology and enhanced to meet new user needs - all without enough regard to overall architecture. The result is the poorly designed structures, poor coding, poor logicand poor documentation of the software systems we are now called on to keep running...” [OSB 90]

performance, a confiabilidade e facilitando posterior manutenção”<sup>14</sup>.

Segundo Pressman, “Reengenharia leva tempo; custa significantes somas em dinheiro; e absorve recursos que poderiam estar ocupados com outras preocupações”<sup>15</sup>. Com isso pode-se ver que a reengenharia de um sistema deve ser feita com uma boa estratégia, um bom processo, de forma a garantir bons resultados.

Alcântara [ALC 02, p. 9] cita algumas vantagens do processo de reengenharia como o “aumento da manutenibilidade do software” e a “valorização do sistema que sofre o processo de reengenharia”, porém também chama a atenção para algumas preocupações que se deve ter quanto à forma de implementação da reengenharia, citando aspectos como “implantação gradual e integração ortogonal”, um para melhorar a adaptação dos usuários do sistema antigo frente ao novo e outro para evitar problemas de compatibilidade entre as duas versões do sistema.

### 2.2.1 Modelos de processo de reengenharia de software

Pressman, em [PRE 00, p. 805-809], apresenta um modelo de processo de reengenharia baseado em seis passos:



Figura 2: Modelo de processo de reengenharia de software [PRE 00, p. 807]

<sup>14</sup>Adaptado de [PRE 00, p. 799]: “You’ll need to rebuilt it. You’ll create a product with added functionality, better performance and reliability, and improved maintainability”

<sup>15</sup>“Reengineering takes time; it costs significant amounts of money; and it absorbs resources that might be otherwise occupied on immediate concerns” [PRE 00, p. 805].

**Análise de Inventário.** Um inventário de software nada mais é do que uma lista com alguns detalhes dos softwares de uma organização. Essa lista pode conter informações como tamanho, idade, nível de necessidade da ferramenta para a organização, etc.

Ao se analisar tal inventário pode-se chegar a uma lista dos sistemas que precisam passar por uma reengenharia e até mesmo a quais são mais prioritários, podendo assim se direcionar esforços para os sistemas mais críticos.

**Reestruturação de Documentos.** Para esta atividade existem, segundo Pressman [PRE 00, p. 807], três opções:

1. Se o sistema é estável, sua vida útil está chegando ao fim e poucas mudanças deverão ser feitas e este é apenas mais um dentre vários sistemas a serem analisados, então não é necessário desperdiçar muito tempo criando documentação sobre o sistema antigo, mais vale deixar este passo para trás.
2. Se o sistema não está sendo completamente mudado, mas apenas parte dele realmente está sendo trabalhada, não é necessário se fazer a documentação de todo ele, gera-se documentação do que é trabalhado.
3. Se o sistema é crítico para o negócio como um todo, então é altamente recomendado que se gere documentação para todo o sistema legado.

**Engenharia Reversa.** Segundo Pressman [PRE 00, p. 807], a engenharia reversa teve origem na indústria de hardware, onde empresas desmontavam produtos de concorrentes a fim de entender os seus “segredos”. Porém, sem a documentação de projeto dos produtos era extremamente difícil entender o seu funcionamento, para isso era necessário se fazer uma análise detalhada e gerar a documentação necessária para a compreensão do produto.

Em essência, “a engenharia reversa, parte de um sistema existente, analisando-o de forma a identificar seus componentes e as inter-relações dos mesmos. A engenharia reversa tem como um dos principais benefícios, a obtenção de resultados na recuperação de informações e estruturas úteis” [ALC 02, p. 8].

A engenharia reversa de softwares segue a mesma linha, a diferença é que em geral o sistema analisado não foi criado por um concorrente, mas pela própria companhia, que dispõe apenas do código fonte e de pouca ou nenhuma documentação. Alcântara cita que “para o software a engenharia reversa é o processo de analisar um programa, num esforço

para criar uma representação do programa em nível de abstração maior do que o código fonte, sendo um processo de recuperação de projeto” [ALC 02, p. 7].

**Reestruturação de Código.** Esta atividade é utilizada em trechos de códigos especialmente difíceis de se entender. Nestes casos, o código é cuidadosamente analisado e então reescrito ou adaptado de forma a ficar mais compreensível e sua documentação é atualizada.

**Reestruturação de Dados.** Um programa com uma arquitetura de dados fraca é bastante difícil de se adaptar. A reestruturação de dados consiste em se fazer uma engenharia reversa da estrutura de dados para, posteriormente, se adaptar ou recriar toda a estrutura de forma a gerar uma arquitetura mais sólida. É claro que normalmente alterações no nível de dados resultarão na necessidade de alterações no código que os manipula.

**Engenharia Progressiva.** Consiste em uma atividade similar à tradicional engenharia de software, onde os dados iniciais provém da análise de um sistema legado e o objetivo não é criar um sistema novo ou simplesmente dar uma cara nova ao sistema legado, mas sim adaptar o antigo sistema à realidade atual dos usuários adicionando novas funcionalidades e atendendo a novos requisitos.

Esses passos não precisam estar necessariamente na seqüência apresentada na figura 2, em algumas situações pode ser necessário por exemplo, se fazer a engenharia reversa antes da reestruturação de documentos, ou a reestruturação dos dados antes da reestruturação do código.

Outra característica do modelo apresentado na figura 2 é que ele é cíclico, ou seja, alguns passos poderão ser refeitos e um processo de reengenharia em particular pode terminar em qualquer passo.

Outro modelo de processo de reengenharia é proposto por Lemos [LEM ] enfocando não apenas a reengenharia do software em si, mas sim este processo dentro do contexto de BPR. Para Lemos a metodologia pode ser baseada em 7 etapas<sup>16</sup>:

1. **Definição dos objetivos da reengenharia e do negócio.** Nesta fase é fundamental o envolvimento da alta administração para que os objetivos táticos/estratégicos sejam bem disseminados no projeto de reengenharia.
2. **Constituição da equipe de reengenharia.** Deve ser uma equipa integrada e

---

<sup>16</sup>Todos os passos aqui descritos estão baseados em [LEM , p. 1-3].

multidisciplinar composta por programadores, usuários e gestores de negócio e do departamento de Sistemas de Informação. É essencial que os utilizadores finais estejam bem representados para que todas as necessidades de informação sejam bem definidas e representativas da realidade operacional.

3. **Análise do Sistema Legado.** É uma das fases mais longas do projeto, dado que a maior parte das parte-se de uma fraca documentação e mapeamento dos SI existentes na empresa. É uma etapa importante, visto que ela é que irá determinar e condicionar a execução da reengenharia.
4. **Especificação dos novos requisitos do sistema.** Identificados os objetivos e efetuado o re-mapeamento dos programas, dá-se início à especificação funcional e técnica do novo sistema com os elementos funcionais e técnicos do projecto.
5. **Definição do processo de implementação.** O processo de implementação pode ser efectuado segundo duas formas distintas: através de um processo em bloco, onde todas as mudanças ao sistema legado são postas em prática e validadas de uma só vez; ou através de um processo incremental onde à medida que as mudanças são desenvolvidas estas são validadas e colocadas em produção. A escolha do processo de implementação depende da especificidade e risco do projecto em causa.
6. **Desenvolvimento e testes.** É uma etapa crítica na medida em que é difícil provar que o novo sistema é funcionalmente equivalente ao software legado.
7. **Formação:** a formação é uma peça fundamental para a consolidação do projeto a todos aqueles que irão utilizar o novo sistema.

Lemos também ressalta a importância dos processos de engenharia reversa e engenharia progressiva, por ele chamados de recuperação e redesevolvimento, respectivamente. Neste modelo de processo tais atividades estão incluídas nos passos 3, análise do sistema legado, e 6, desenvolvimento e testes.

### 2.2.2 Razões que levam ao fracasso de uma reengenharia

Segundo um relatório técnico do SEI [BER 99], estes são os dez principais motivos que levam projetos de reengenharia ao fracasso:

1. **A organização inadvertidamente adota uma estratégia de reengenharia defeituosa ou incompleta.**

2. **A organização faz uso inapropriado de consultoria externa ou serviços terceirizados.** Serviços terceirizados podem ser de extrema utilidade para resolução de questões técnicas, obtenção de conhecimento do negócio, etc. Porém, em geral o trabalho de terceiros deve ser monitorado mais de perto para evitar a ocorrência de problemas. Outro problema possível é a não contratação de um consultor ou serviço terceirizado quando necessário, o que pode ser tão determinante para o fracasso quanto uma contratação mal feita.
3. **A força de trabalho está presa a tecnologias antigas sem um programa de treinamento adequado.** É comum que uma reengenharia venha acompanhada de mudanças de tecnologia, metodologias, equipamentos, paradigmas e até mesmo vocabulários. Para tanto é necessário um bom programa de treinamento para preparar e motivar o pessoal da equipe para a fazer a migração.
4. **A organização não possui o sistema legado sob controle.** Conhecimento das mudanças feitas no sistema, dos tipos de requisitos alterados, do grau de dificuldade e de importância das alterações e existência de métricas sobre as atividades de manutenção são características de um sistema sob controle. Um exemplo de situação em que a reengenharia não fracassa por este motivo é quando se troca as métricas e o histórico de manutenção por adições sobre esforço, custo e tempo necessários para realizar alguma alteração.
5. **Há pouco levantamento e validação de requisitos.** Uma causa de falhas em reengenharia é o descaso com a análise dos requisitos de um sistema legado. Existem dificuldades nesta tarefa pois o sistema, em geral, ajuda pouco, já que normalmente foi baseado em requisitos antigos e já não completamente válidos para o negócio. E deve-se direcionar um bom esforço no sentido de levantar os novos requisitos para o sistema.
6. **A arquitetura do software não é considerada um fator primordial.** Uma avaliação apropriada da arquitetura do software irá servir de base para a definição da abordagem da reengenharia. Se a arquitetura legada for considerada inviável, deve-se construir um novo sistema completo, porém, se a arquitetura for considerada parcial ou totalmente viável, deve-se partir daquilo que pode ser aproveitado, desde que a compreensão da arquitetura seja profunda, de outro modo o melhor é construir uma nova.
7. **Não existe noção de um processo de reengenharia.** A falta de um processo

documentado para a realização da reengenharia aumentará muito as chances de o projeto virar um fracasso. Sem um bom processo dificilmente uma boa equipe com boas ferramentas de desenvolvimento produzirá um bom resultado.

8. **Existe um planejamento inadequado ou pouca dedicação em seguir o planejamento.** Um plano pode ser feito mas não posto no papel, ficando apenas na cabeça de algumas pessoas-chave. O planejamento pode ser feito mas não passado adiante para todos os interessados. Também pode ocorrer de o plano ser incompleto, ou de não se ter recursos para implementá-lo ou ainda o time pode não se comprometer em seguir o planejamento. Em qualquer um desses casos os riscos de falha crescem bastante.
9. **O gerenciamento carece de comprometimento a longo prazo.** Um gerenciamento cuidadoso de um projeto de reengenharia é muito importante para diminuir a chance de ocorrência de erros que, se achados apenas em estágios mais avançados do desenvolvimento, serão extremamente caros para consertar. Por essa razão deve-se evitar que o gerente de um projeto de reengenharia tenha distrações como outros projetos.
10. **O gerenciamento pré-determina questões técnicas.** Quando um membro da gerência indica tecnologias usadas, custos, agenda e questões de performance sem a intervenção de pessoal da área técnica, a chance de tais determinações falharem é bastante grande, podendo prejudicar em muito o projeto todo.

### 2.2.3 Engenharia Reversa

Um dos pontos centrais de qualquer processo de reengenharia é a análise do sistema legado via engenharia reversa. Este processo pode parecer simples, porém nem sempre é possível se chegar a resultados satisfatórios.

“Engenharia reversa pode extrair informações de projeto do código fonte, mas o nível de abstração, a completude da documentação, o grau em que ferramentas e analistas humanos trabalham juntos, e a direcionalidade do processo são muito variáveis”<sup>17</sup>.

O nível de abstração indica o tipo de informações que podem ser retiradas do código fonte. O ideal é que o nível de abstração seja o mais alto possível, pois quanto maior o

---

<sup>17</sup>“Reverse engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and a human analyst work together, and the directionality of the process are highly variable” [cas 88].



nível de abstração, mais compreensível se tornará o sistema.

A completude indica o quão abrangente é o processo de engenharia reversa. Quanto mais abrangente ele for, mais difícil será manter um nível de abstração alto, pois isso demandará muito trabalho. Para solucionar isso pode-se contar com a interatividade, que indica o quanto os analistas humanos estão interagindo com ferramentas automatizadas para agilizar o processo e torná-lo mais efetivo. Para Pressman [PRE 00, p. 809], na maioria dos casos uma alta abstração só não reduzirá a completude se o nível de interatividade for alto.

A direcionalidade pode se apresentar de duas formas, de uma via, em que as informações da engenharia reversa são encaminhadas para um engenheiro de software com o fim de fazer a manutenção do sistema, ou de duas vias, em que a informação vai para um processo de reengenharia, com o fim de reestruturar ou reconstruir o sistema antigo.

O processo de engenharia reversa inicia-se com o código desestruturado ou simplesmente não documentado. Quando necessário, o primeiro passo deve ser uma estruturação do código a fim de facilitar sua compreensão. O passo seguinte é o que Pressman definiu como “o âmago da engenharia reversa” [PRE 00, p. 810], a extração de abstrações. Esta atividade consiste em se extrair do código informações significantes sobre o processamento, a interface com o usuário e a base de dados do sistema.

Para se criar a abstração do processamento do sistema, o código pode ser analisado em diferentes níveis, partindo do mais alto, o sistema e seus módulos, e baixando para compreensão de componentes e padrões. As abstrações dos níveis mais altos são mais simples de serem feitas, porém, quando se entra nas abstrações de componentes e padrões a análise começa a ficar mais complexa, pois, em geral, eles são partes de código genéricos que devem ser entendidos separadamente e dentro do contexto do sistema.

A abstração da base de dados pode ser feita em dois níveis, primeiramente identificando-se os conceitos armazenados separadamente. Em seguida, parte-se para a identificação das ligações entre os conceitos com o fim de se ter uma visão geral da estrutura da base de dados para poder, enfim, remodelá-la em um paradigma moderno de bancos de dados.

A engenharia reversa de interfaces com o usuário pode ser feita, segundo Merlo [MER 93], a partir de três questões básicas:

- Quais são as ações básicas que a interface deve processar?
- Quais as descrições dos comportamentos de resposta do sistema a essas ações?

- O que é pretendido em uma interface substituta?

## 3 *Rational Unified Process (RUP)*

“O Rational Unified Process® é um processo de engenharia de software. Ele provê uma abordagem disciplinada para determinação de tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de software de alta qualidade que concilie-se com as necessidades do usuário final, dentro de prazo e orçamento previsíveis” [RAT 98, p. 1]. Além disso, o RUP “é um guia de como se usar efetivamente o Unified Modeling Language (UML). O UML é uma linguagem padrão que permite comunicar claramente requisitos, arquitetura e projeto de software” [RAT 98, p. 1].

“O RUP é um produto proprietário desenvolvido e mantido pela Rational Software Corporation, recentemente comprada pela IBM” [SCH 04, p. 3].

Segundo Paula Filho [dPPF 03, p. 20], o RUP é um produto comercial baseado no Unified Process (UP). O UP foi proposto pelos mesmo autores da UML e a utiliza como notação de diversos modelos usados no processo. O UP descende de métodos anteriores propostos por Booch, Jacobson e Rumbaugh, os pais de UML e UP. As principais características de UP, segundo [dPPF 03, p. 20] são:

- é dirigido por casos de uso;
- é centrado na arquitetura;
- é iterativo incremental.

As diferenças entre o UP e o RUP estão em sua estrutura de fluxos de trabalho, sendo que o RUP possui alguns fluxos adicionais, e no fato de que o UP é descrito em um livro, enquanto o RUP é vendido como uma base de conhecimento em hipermídia com milhares de arquivos, o que evidencia seu maior nível de detalhamento.

O RUP descreve como fazer um desenvolvimento efetivo de software fazendo-se valer das chamadas “boas práticas” de desenvolvimento. Em [RAT 98, p. 1] estão descritas algumas dessas boas práticas e a forma como RUP as trabalha<sup>1</sup>:

**Desenvolvimento iterativo de software.** Atualmente os sistemas desenvolvidos são sofisticados demais para se definir o problema, analisá-lo e projetar e construir a solução de forma seqüencial em apenas uma iteração. Uma abordagem adequada é via iterações, onde o conhecimento sobre o problema vai crescendo, os refinamentos são sendo feitos e a solução vai crescendo de forma incremental. O RUP suporta uma abordagem iterativa em que, a cada ciclo, o item de maior risco é atacado, de forma que os riscos são significativamente reduzidos, já que os maiores problemas são encontrados e sanados já no início.

**Gerenciamento de requisitos.** O RUP descreve como obter, organizar e documentar requisitos. A adoção de casos de uso (veja a seção 3.1.3.1, página 41) provou ser um excelente modo de capturar os requisitos funcionais e de assegurar que estes guiarão o desenvolvimento do software, garantindo que o resultado final atenda às necessidades do usuário.

**Uso de arquiteturas baseadas em componentes.** Componentes são subsistemas complexos que desempenham uma função bem definida. RUP provê uma abordagem sistemática para se definir uma arquitetura usando componentes, sejam eles padrões bem definidos ou novos componentes criados pela própria equipe.

**Modelagem visual de software.** Abstrações visuais ajudam a comunicar diferentes aspectos de um software como a interação entre componentes ou elementos. A linguagem UML é usada em RUP como linguagem de modelagem visual.

**Verificação da qualidade do software.** Aplicações não confiáveis ou com baixo desempenho dificilmente terão boa aceitação no mercado, por isso RUP ajuda a planejar, projetar, implementar, executar e avaliar os testes aplicados em todo o desenvolvimento. As avaliações de qualidade estão embutidas em todo o processo, em todas as atividades, envolvendo todos os participantes e usando critérios e medidas objetivas e não sendo vistas como uma atividade a parte, realizada depois de todo o processo.

**Controle das mudanças no software.** Em uma atividade como o desenvolvimento de softwares, onde mudanças são inevitáveis, a habilidade de gerenciar mudanças

---

<sup>1</sup>os textos dos itens abaixo foram baseados em [RAT 98, p. 2]

e torna-las aceitáveis, rastreando todas as suas conseqüências, é crucial. O RUP descreve como controlar, rastrear e monitorar as mudanças. Ele também auxilia a estabelecer divisões de forma que se pode garantir que uma divisão estará isolada de outra no tocante às alterações.

O processo RUP pode ser descrito em duas dimensões, conforme mostrado na figura 3:

- “O eixo horizontal representa o tempo e mostra o aspecto dinâmico do processo, e é expresso em termos de ciclos, fases, iterações e marcos” [RAT 98, p. 3].
- “O eixo vertical representa o aspecto estático do processo: como ele é descrito em termos de atividades, artefatos, trabalhadores e fluxos de trabalho” [RAT 98, p. 3].

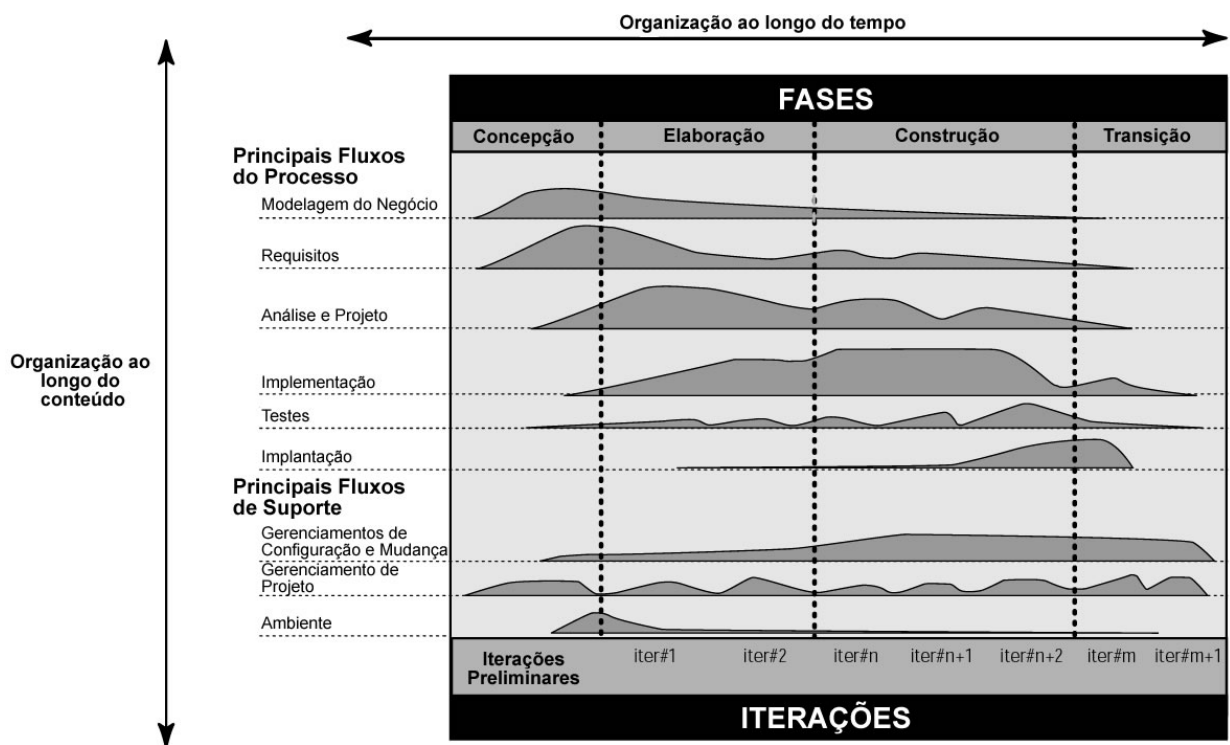


Figura 3: Estrutura geral do RUP em duas dimensões [RAT 98, p. 3]

### 3.1 Principais fluxos de trabalho do processo

Os fluxos de trabalho “representam uma divisão de todos os trabalhadores e atividades em grupos lógicos” [RAT 98, p. 10]. Em geral, eles estão presentes em todas as fases do

desenvolvimento, com maior ou menor ênfase, dependendo da fase, o que pode ser visto na figura 3, página 36.

### 3.1.1 Modelagem do Negócio

Segundo [RAT 98, p. 10], existe um grande déficit de comunicação entre as comunidades de engenharia de software e de negócios, o que leva a uma situação em que a documentação gerada pelo pessoal de engenharia de software não é exatamente aquilo sugerido pelo pessoal de negócios.

Uma forma de solucionar esse problema é estabelecendo-se uma linguagem comum às duas comunidades, o que RUP fez com os casos de uso<sup>2</sup>. “Isso garante um entendimento comum de todos os interessados sobre quais processos de negócio precisam ser suportados na organização” [RAT 98, p. 10].

### 3.1.2 Levantamento de Requisitos

“Requisitos são uma descrição das necessidades ou dos desejos para um produto” [LAR 00, p.60]. “O fluxo de requisitos reúne as atividades que visam a obter o enunciado completo, claro e preciso dos requisitos de um produto de software” [dPPF 03, p. 87]. Segundo Larman [LAR 00, p. 60], “o desafio é definir os requisitos de maneira não-ambígua, de modo que os riscos sejam identificados e não aconteçam surpresas quando o produto for finalmente liberado”.

Para realizar o levantamento de requisitos, Wazlawick [WAZ 04, p. 34-36] indica que o analista pode produzir alguns artefatos como os seguintes:

- **Visão geral do sistema ou sumário executivo.** Texto corrido, sem necessidade de estrutura especial, que descreve as principais idéias do cliente sobre o sistema. Para Wazlawick [WAZ 04, p. 37], não precisa ser longo, em geral, em mais do que duas páginas já se está incluindo detalhes desnecessários.
- **Requisitos funcionais e não funcionais.** Registram, respectivamente, o que o sistema deve fazer e como essas coisas devem ser feitas. Wazlawick [WAZ 04, p. 38] e Paula Filho [dPPF 03, p. 87] sugerem que esta atividade deve ser realizada pela equipe de desenvolvimento juntamente com clientes e usuários.

---

<sup>2</sup>Casos de uso são detalhados na seção 3.1.3.1, página 41.

- **Glossário.** Documento em que são definidos os termos técnicos do domínio da aplicação. É importante pois muitas vezes uma mesma palavra pode levar a diferentes interpretações em diferentes situações e duas palavras que para o senso comum são consideradas sinônimos podem não sê-lo no domínio da aplicação.
- **Análise de riscos e seu controle.** Documento com a análise dos principais riscos no desenvolvimento. Isto é importante para que se tente abordar tais itens o mais cedo possível nos ciclos iterativos. “Muitas vezes não é possível prever se um risco pode ser neutralizado logo no início do desenvolvimento, mas pode-se pelo menos prever a existência de mecanismos de controle” [WAZ 04, p. 36].
- **Protótipos e provas.** Quando for necessário se verificar e esclarecer alguns requisitos, pode-se produzir um protótipo do tipo “*throw-away*”, ou seja, cujo código não será usado no desenvolvimento, mas jogado fora assim que as dúvidas forem elucidadas.

### 3.1.2.1 Requisitos

O levantamento de requisitos pode ser feito, segundo Wazlawick [WAZ 04, p. 38], em uma reunião do tipo *brainstorm* (tempestade cerebral) com a equipe de desenvolvimento e usuários e clientes.

Os requisitos podem ser classificados em duas categorias:

- **Requisitos funcionais.** Listagem de tudo o que o sistema deve fazer.
- **Requisitos não funcionais.** Restrições sobre como o sistema deve realizar os requisitos funcionais.

Os requisitos funcionais ainda podem ser divididos em dois grupos:

- **Requisitos funcionais evidentes.** Aqueles que são efetuados com o conhecimento do usuário, ou seja, eventos e respostas do sistema.
- **Requisitos funcionais ocultos.** Efetuados sem o conhecimento explícito do cliente.

Já os requisitos não funcionais “podem ser classificados como obrigatórios e desejados, isto é, aqueles que devem ser obtidos de qualquer maneira e aqueles que podem ser

obtidos caso isso não cause maiores transtornos no processo de desenvolvimento” [WAZ 04, p. 38-39].

Em geral, os requisitos não funcionais são associados a uma função, porém também existem requisitos não funcionais gerais para o sistema, estes são chamados de requisitos *suplementares*.

### 3.1.2.2 Organização dos Requisitos

Depois de identificados, os requisitos devem ser organizados em grupos correlacionados para, posteriormente, serem abordados nos ciclos iterativos. Como é indicado por Wazlawick [WAZ 04, p. 44-45], os requisitos podem ser agrupados em *casos de uso*, *conceitos* e *consultas*.

#### Organização dos Requisitos em Casos de uso

Casos de uso representam os principais processos de negócio da organização e abrangem, em geral, um considerável número de requisitos funcionais.

Inicialmente deve-se partir de uma lista dos casos de uso, cuja elaboração já começa na modelagem do negócio, contendo seus atores, uma breve descrição e a lista dos requisitos correlacionados.

Alguns aspectos que se deve ter em mente ao identificar os casos de uso são:

- “Um caso de uso deve ser monossessão, isto é, executado em uma única interação e não se estendendo por vários dias” [WAZ 04, p. 48].
- Um caso de uso deve ser interativo, ou seja, informações devem fluir tanto para dentro quanto para fora do sistema. Dessa forma considera-se que um cadastro não é um caso de uso, já que as informações apenas entram no sistema, bem como uma consulta, em que as informações apenas saem do sistema.

#### Organização dos Requisitos em Função de Conceitos.

São informações tratadas pelo sistema. Em geral, os conceitos sofrem operações de manutenção, que podem ser inserção, alteração, remoção e consulta. Tais operações são simples demais para se tornarem casos de uso e, muitas vezes aparecem como parte de um caso de uso maior.

A correta identificação dos conceitos e operações de manutenção pode ser facilitada, segundo Wazlawick [WAZ 04, p. 49], com a construção de um modelo conceitual inicial,



um diagrama que contém os conceitos e associações que constituem a informação a ser armazenada pelo sistema, como no exemplo da figura 4.

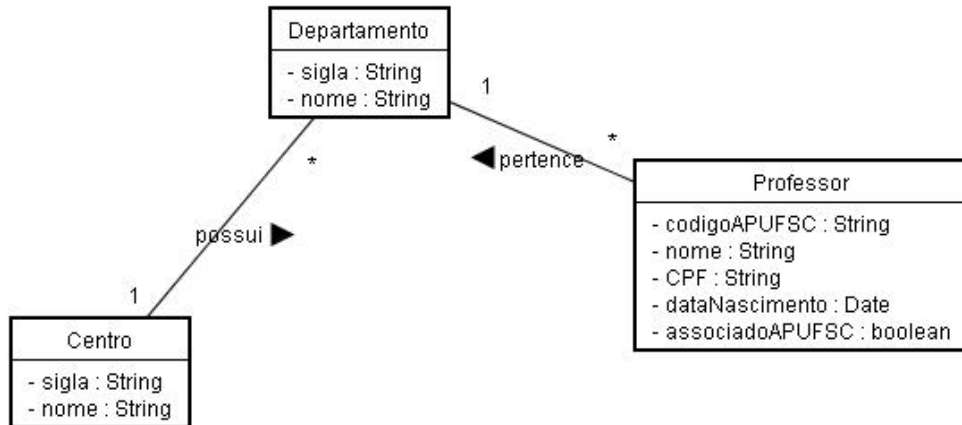


Figura 4: Modelo conceitual com informações sobre a relação entre professores, departamentos e centros.

Assim que os conceitos são identificados, pode-se criar uma tabela indicando os conceitos, as operações de manutenção aplicáveis a cada conceito e observações pertinentes. A tabela 1 mostra um exemplo em que as operações são identificadas por suas letras iniciais: (I)ncluir, (A)lterar, (E)xcluir e (C)onsultar.

Conceito	I	A	E	C	Observações
Professor	X	X		X	
Centro	X	X	X	X	Somente pode ser excluído se não houver professor algum associado.
Departamento	X	X	X	X	Somente pode ser excluído se não houver centro algum associado.

Tabela 1: Exemplo de tabela de conceitos.

### Organização dos Requisitos em Consultas.

Abrangem, segundo Wazlawick [WAZ 04, p. 45], tanto as consultas apresentadas em tela quanto aquelas feitas para geração de relatórios. Consulta é qualquer acesso à informação armazenada no sistema que não gere qualquer tipo de alteração nos dados.

Aqui pode-se criar uma listagem das consultas podendo-se restringir essa lista àquelas consultas mais complexas, que reúnem dados de mais de um conceito.

### 3.1.3 Análise

O fluxo de análise, segundo Paula Filho [dPPF 03, p. 120], visa aos seguintes objetivos:

- Modelar de forma precisa os conceitos relevantes do domínio do problema.
- Verificar a qualidade dos requisitos obtidos através do fluxo de requisitos.
- Detalhar esses requisitos o suficiente para que atinjam o nível de detalhe adequado aos desenvolvedores.

Para se alcançar tais objetivos a análise comporta, segundo Wazlawick [WAZ 04, p. 60], três atividades:

- Expansão dos casos de uso e determinação dos eventos de sistema.
- Construção do modelo conceitual.
- Elaboração dos contratos das operações de sistema.

#### 3.1.3.1 Expansão dos Casos de Uso

Atividade equivalente, segundo Wazlawick [WAZ 04, p. 61], a um aprofundamento de requisitos. A expansão dos casos de uso começa com um exame detalhado do processo abordado, descrevendo-se, a princípio, a sequência de passos normal do processo, aquela em que tudo ocorre perfeitamente. A essa sequência denomina-se *fluxo principal*.

Assim que o fluxo principal está descrito, passa-se a uma análise de cada passo deste a fim de se encontrar tudo o que possa dar errado. Para cada possível exceção é escrita uma *seqüência alternativa*.

O foco dos fluxos principal e alternativos deve estar nas trocas de informação, desconsiderando-se, aqui na análise, questões de interface e tecnologia. Por essa razão, como pode ser visto em Wazlawick [WAZ 04, p. 63], esse tipo de caso de uso é chamado *essencial*.

Além das seções do fluxo principal e das seqüências alternativas, um caso de uso pode conter outras como:

- **Atores.** Uma lista dos atores, ou seja, pessoas ou outros sistemas que interagem com o sistema através do caso de uso descrito.

- **Interessados** (Stakeholders). Algumas vezes, além dos atores existem outros interessados nos resultados de um caso de uso, como o cliente de uma loja durante um caso de uso de venda de um produto.
- **Pré-condições**. Fatos considerados verdadeiros antes do início do caso de uso, ou seja, não devem ser testados dentro do caso de uso.
- **Pós-condições**. Estabelecem tudo o que deve ser verdade ao final de uma execução de sucesso de um caso de uso.
- **Requisitos correlacionados**. Relação dos requisitos que possuem alguma relação com o caso de uso. Isso pode ajudar os analistas a descobrir requisitos ainda não abordados.
- **Questões em aberto**. Quando o analista trabalha sem a presença do cliente algumas questões podem não estar claras, então pode-se listá-las para posterior consulta ao cliente. Deve-se tomar cuidado para não deixar questões em aberto até depois do final da análise.

### 3.1.3.2 Construção do Modelo Conceitual

“O modelo conceitual deve descrever a informação que o sistema vai gerenciar” [WAZ 04, p. 102]. Sua ênfase deve estar na compreensão da informação representada, e não na sua representação física, ou seja, forma de representação e organização dos dados.

O objetivo na construção do modelo conceitual é definir “quais são os elementos de informação tratados pelo sistema para que mais adiante se possa mostrar ainda como essa informação é transformada pelo sistema a partir de diferentes requisições do usuário (operações de sistema)” [WAZ 04, p. 102].

Portanto, um modelo conceitual deve representar apenas o mundo exterior ao sistema, sem dar atenção a questões de arquitetura do sistema. Tal abordagem deve ser feita posteriormente, no fluxo de Projeto.

Um modelo conceitual pode ser construído utilizando-se um diagrama de classes de UML. Dessa forma, segundo Wazlawick [WAZ 04, p. 104] são três os principais elementos para se representar o modelo conceitual:

- **Conceitos**. São a representação da informação complexa, representados, no diagrama, por classes. Na figura 4, *Professor*, *Centro* e *Departamento* são exemplos de conceitos.

- **Atributos.** Informações alfanuméricas diretamente ligadas aos conceitos, como *nome* e *sigla* nos atributos Centro e Departamento da figura 4.
- **Associações.** Consistem em um tipo de informação que liga diferentes conceitos entre si, como na figura 4, onde *possui*, que liga um centro a vários departamentos e *pertence*, que liga vários professores a um departamento. A associação possui não somente um nome, mas também uma direção (centro que possui departamentos e não o contrário) e cardinalidades (um centro para vários departamentos).

### 3.1.3.3 Elaboração dos contratos

Contratos são descrições funcionais das operações de sistema e consultas, eles descrevem “o que uma operação se compromete a atingir” [LAR 00, p. 152].

Segundo Wazlawick [WAZ 04, p. 152], os contratos são divididos em dois tipos, de operação de sistema e de consulta, sendo que cada um é construído de uma maneira. Um contrato de operação de sistema deve ter ao menos as seções de pré-condições, pós-condições e exceções, enquanto um contrato de consulta tem apenas duas seções, pré-condições e resultados. Além disso, “cada contrato poderá ter ainda uma seção de objetivos, composta de um texto simples e curto que procura explicar a intenção do usuário ao executar a operação ou consulta” [WAZ 04, p. 153].

As pré-condições, assim como nos casos de uso, definem “o que deve ser verdadeiro na estrutura da informação armazenada para que a operação ou consulta possa ser executada” [WAZ 04, p. 153].

As pós-condições determinam o que será alterado na estrutura da informação armazenada ao final da operação.

Os resultados de consultas descrevem as informações que serão lidas das informações armazenadas.

As exceções ocorrem quando se tenta alterar alguma informação e não se consegue, por isso elas só ocorrem nas operações, já que consultas não alteram dados.

### 3.1.4 Projeto

Para Paula Filho, o fluxo de projeto “tem por objetivo definir uma estrutura implementável para um produto de software que atenda aos requisitos especificados para ele” [dPPF 03, p. 148]. Ainda de acordo com Paula Filho, o projeto de software deve

considerar os seguintes aspectos:

- O atendimento dos requisitos não-funcionais.
- A definição de classes e outros elementos de modelo em nível de detalhe suficiente para a respectiva implementação.
- A decomposição do produto em componentes cuja construção seja relativamente independente.
- A definição adequada e rigorosa das interfaces entre os componentes do produto.
- A documentação das decisões de projeto, de forma que essas possam ser comunicadas e entendidas por quem vier a implementar e manter o produto.
- A reutilização de componentes, mecanismos e outros artefatos para aumentar a produtividade e a confiabilidade.
- O suporte a métodos e ferramentas de geração semi-automática de código.

Para Larman, durante o fluxo de projeto, “é desenvolvida uma solução lógica baseada no paradigma orientado a objetos. O coração desta solução é a criação de **diagramas de interação**, os quais ilustram como os objetos devem se comunicar de maneira a atender os requisitos” [LAR 00, p. 166].

Feitos os diagramas de interação, Larman indica que podem ser criados **diagramas de classe de projeto**, “os quais sumarizam a definição das classes (e interfaces) que devem ser implementadas em software” [LAR 00, p. 166].

#### 3.1.4.1 Diagramas de Interação

“Um diagrama de interação mostra uma interação, consistindo de um grupo de objetos e seus relacionamentos, incluindo as mensagens que devem ser lançadas entre eles” [BOO 98, p. 203].

Existem dois tipos de diagramas de interação:

- Diagrama de sequência, que dá maior ênfase à organização cronológica das mensagens.
- Diagrama de colaboração, enfatiza a organização estrutural dos objetos que enviam e recebem as mensagens.

Neste momento do desenvolvimento o diagrama de colaboração pode ser priorizado por motivos como os apresentados por Larman: “por causa da sua capacidade de expressão excepcional, da sua habilidade para expressar mais informações contextuais e da sua relativa economia de espaço” [LAR 00, p. 173].

Para a construção dos diagramas de colaboração neste ponto deve-se ter os seguintes artefatos prontos:

- Modelo conceitual. Indica os objetos que trocarão mensagens no diagrama de colaboração.
- Contratos das operações do sistema. Indica as responsabilidades e as pós-condições que os diagramas de interação devem satisfazer.
- Casos de uso reais. São casos de uso como os da análise, porém mais detalhados, contendo até informações sobre a interface. Pode auxiliar no levantamento de mensagens não encontradas nos contratos.

#### 3.1.4.2 Diagramas de Classes

É o diagrama mais comum e conhecido de UML e representa o conjunto de classes e interfaces de um sistema e seus relacionamentos. O modelo conceitual (veja a seção 3.1.3.2, página 42) construído na análise é uma versão simplificada de um diagrama de classes.

Um diagrama de classes representa as classes e interfaces do sistema com todos os atributos, métodos e relacionamentos necessários para que o sistema funcione, ao contrário do modelo conceitual, onde os atributos e relacionamentos visavam abstrair a realidade sem se importar com o funcionamento do sistema e os métodos ainda não eram listados.

Os métodos listados no diagrama de classes são encontrados a partir das mensagens passadas nos diagramas de colaboração.

#### 3.1.5 Implementação

A partir do projeto, “são implementados os subsistemas, componentes, códigos fontes, *scripts*, programas executáveis e outros” [MAR 02, p. 121].

Dentre as tarefas a serem realizadas na implementação, Paula Filho [dPPF 03, p. 200] cita as seguintes:

- Planejamento detalhado da implementação das unidades de cada iteração.
- Implementação das classes e outros elementos do modelo de projeto, em unidades de implementação, geralmente constituídos de arquivos de código fonte.
- No caso de sistemas distribuídos, alocação das unidades aos nodos do sistema.
- Verificação das unidades, po meio de revisões, inspeções e testes de unidade.
- Compilação e ligação das unidades.
- Integração das unidades entre si.
- Integração das unidades com componentes reutilizados, adquiridos de terceiros ou reaproveitados de projetos anteriores.
- Integração das unidades com os componentes resultantes das liberações anteriores.

### 3.1.6 Testes

Como pode ser visto em [RAT 98, p. 12], o propósito dos testes está em:

- Verificar a interação entre objetos.
- Verificar a integração apropriada entre todos os componentes de software.
- Verificar que todos os requisitos tenham sido corretamente implementados.
- Identificar defeitos e assegurar que eles serão prioridade no fluxo de implantação.

“O Rational Unified Process propõe uma abordagem iterativa, o que significa que você testará ao longo do desenvolvimento. Isso permite encontrar defeitos o mais cedo possível, o que reduz radicalmente o custo de consertar os defeitos” [RAT 98, p.12].

Os testes devem avaliar três dimensões de qualidade: confiança, funcionalidade e performance.

### 3.1.7 Implantação

O propósito neste fluxo é produzir uma versão do produto e entregá-la ao usuário final. Segundo [RAT 98, p. 13], algumas das atividades deste fluxo são:

- Produção de versões externas do software.
- Embalagem do software.
- Distribuição do software.
- Instalação do software.
- Ajuda e assistência aos usuários.
- Migração dos dados da versão existente.

## 3.2 Principais fluxos de trabalho de suporte

### 3.2.1 Gerenciamento de configuração e mudança

Fluxo dedicado a controlar os diversos artefatos produzidos durante o desenvolvimento. Este controle ajuda, como pode ser visto em [RAT 98, p. 13], a evitar problemas e conflitos como os seguintes:

- **Atualização simultânea.** Ocorre quando duas pessoas alteram o mesmo artefato ao mesmo tempo, de forma que, quando o último enviar suas modificações, as alterações feitas pelo primeiro serão perdidas.
- **Notificação limitada.** Um problema é resolvido ou uma alteração é realizada e algumas pessoas da equipe não são notificadas.
- **Múltiplas versões.** Quando um problema é encontrado em uma versão do sistema pode-se precisar propagar sua correção para as demais versões, o que pode causar problemas se as correções não forem bem gerenciadas.

### 3.2.2 Gerenciamento do projeto

Gerenciamento de projeto consistem em determinar rumos para o projeto partindo-se de objetivos divergentes, gerenciar riscos e levar o projeto à liberação de um produto que vá de encontro com as necessidades de clientes e usuários.



### 3.2.3 Ambiente

“O propósito do fluxo de ambiente é prover para a organização de desenvolvimento os ambientes de desenvolvimento de software - processos e ferramentas - necessários para suportar o time de desenvolvimento” [RAT 98, p. 14].

## 3.3 Fases

O ciclo de desenvolvimento do software é dividido em ciclos menores, ou iterações, sendo que cada iteração trabalha com uma nova geração do produto. O ciclo de desenvolvimento também é dividido em quatro fases consecutivas, concepção, elaboração, construção e transição, onde cada uma é constituída de uma ou mais iterações.

Cada fase do desenvolvimento é concluída com um marco bem definido, “um ponto em que certas decisões críticas devem ser feitas, e então objetivos chave deverão ter sido alcançados” [RAT 98, p. 3].

### 3.3.1 Concepção

“A fase de concepção consiste em uma etapa onde o analista vai buscar as primeiras informações sobre o sistema a ser desenvolvido. Nesta etapa, assume-se pouco conhecimento do analista sobre o sistema e uma grande interação com o usuário e cliente” [WAZ 04, p. 32]

“Nesta fase se define o escopo do projeto e sua viabilidade” [SCH 04, p. 9]. Segundo [RAT 98, p. 4], para se chegar a isso deve-se identificar todas as entidades externas com as quais o sistema irá interagir (atores) e definir a natureza dessas interações em alto nível. Esta atividade envolve a identificação dos casos de uso (veja a seção 3.1.3.1, página 41) e a descrição dos mais significantes.

Para [RAT 98], os principais resultados da fase de concepção são:

- Um documento de visão, onde tem-se uma visão geral dos principais requisitos do projeto, as características chave e as principais limitações.
- Um modelo inicial de casos de uso.
- Um glossário inicial do projeto (veja a seção 3.1.2, página 38).

- Um case inicial do negócio, o qual inclui o contexto do negócio, critério de sucesso e previsão financeira.
- Uma avaliação inicial de riscos.
- Um plano do negócio, mostrando fases e interações.
- Um modelo de negócio, se necessário.
- Um ou vários protótipos (veja a seção 3.1.2, página 38).

Além desses resultados, ao final da concepção encontra-se o primeiro marco, os objetivos do ciclo de desenvolvimento.

O documento da Rational [RAT 98, p. 4] ainda enumera os critérios de avaliação da concepção:

- A conformidade dos stakeholders (interessados) com a definição do escopo e dos custos/cronograma estimados.
- Entendimento dos requisitos como evidência pela fidelidade dos casos de uso primários.
- Credibilidade de custos/cronograma estimados, das prioridades, dos riscos e do processo de desenvolvimento.
- Profundidade e largura de qualquer protótipo arquitetural que foi desenvolvido.
- Gastos atuais e os gastos planejados.

### 3.3.2 Elaboração

“O propósito da fase de elaboração é analisar o domínio do problema, estabelecer uma sólida fundação da arquitetura, desenvolver o plano de projeto e eliminar os elementos de maior risco do projeto” [RAT 98, p. 4]. Tais objetivos devem ser alcançados através de uma visão bastante abrangente, porém, pouco profunda do sistema. As decisões sobre a arquitetura devem ser tomadas tendo-se um bom entendimento do sistema como um todo, ou seja, de seu escopo, suas principais funcionalidades e requisitos.

Segunda [RAT 98, p. 4], esta pode ser considerada a fase mais crítica das quatro que compõem o RUP. Ao final o projeto passa pela sua mais importante avaliação: o projeto será levado adiante?

“Na fase de elaboração, um protótipo executável da arquitetura é construído em uma ou mais iterações, dependendo do escopo, tamanho, risco e inovação do projeto” [RAT 98, p.5]. Este protótipo deve contemplar ao menos os principais casos de uso identificados na concepção, que geralmente contém os maiores pontos de risco.

Na fase de elaboração o marco é a definição da arquitetura, mas os resultados também abrangem:

- Aprimoramento do modelo de casos de uso.
- Requisitos suplementares: requisitos não funcionais e requisitos não associados a casos de uso específicos (veja seção 3.1.2, página 37).
- Descrição da arquitetura do software.
- Lista de riscos e case do negócio revisado.
- Plano de desenvolvimento para todo o projeto, mostrando as iterações e critérios de avaliação para cada iteração.
- Caso de desenvolvimento atualizado, especificando o processo a ser usado.
- Manual de usuário preliminar.

Para Wazlawick [WAZ 04, p. 33], algumas perguntas são importantes de se responder a fim de se definir o futuro do projeto: O projeto é realizável? A equipe de desenvolvimento tem condições de realizar este projeto? O cliente tem dinheiro para pagar o desenvolvimento? Há tempo disponível? O que vale mais a pena, comprar um sistema pronto ou construir um novo?

Já em [RAT 98, p. 5], é sugerido que a avaliação da fase de elaboração pode ser feita com base nas seguintes questões:

- A visão do projeto é estável?
- A arquitetura é estável?
- A demonstração executável mostra que os elementos de maior risco têm sido endereçados e resolvidos com confiança?
- O plano para a fase de construção está suficientemente detalhado e preciso? Este plano está de acordo com o que foi estimado?

- Todos os stakeholders concordam que a atual visão pode ser alcançada se o plano corrente for executado para desenvolver o sistema completo, no contexto atual da arquitetura?
- Os gastos atuais e os gastos planejados são aceitáveis?

Se qualquer questão dentre as apresentadas falhar, deve-se abortar o projeto ou então pode-se reconsiderar as decisões já tomadas.

### 3.3.3 Construção

Na fase de construção busca-se completar o desenvolvimento do sistema implementando e integrando todos os componentes e as características do sistema e realizando testes profundos em tudo o que for desenvolvido. Esta fase é “um processo de manufatura, onde a ênfase está em gerenciar recursos e controlar operações para otimizar custos, agenda e qualidade” [RAT 98, p. 6]. Aqui ocorre, segundo [RAT 98], uma passagem do desenvolvimento de propriedade intelectual ocorrido nas fases de concepção e elaboração, para o desenvolvimento de produtos nas fases de construção e transição.

Os resultados esperados ao final da construção são os seguintes;

- Software integrado com as plataformas adequadas.
- Os manuais de usuário.
- Uma descrição da versão atual.

Ao final da construção aparece o terceiro marco no desenvolvimento, a capacidade inicial de operação. Neste ponto deve-se decidir se o sistema está pronto para ir a uso sem expor o projeto a altos riscos. Para se fazer essa avaliação pode-se partir dos seguintes questionamentos<sup>3</sup>:

- O software está em uma versão estável e madura o bastante para ser utilizada pelos usuários finais?
- Todos os interessados (stakeholders) estão prontos para a transição do sistema para a comunidade de usuários?
- Os gastos atuais, frente aos gastos planejados, ainda são aceitáveis?

---

<sup>3</sup>Questões retiradas de [RAT 98, p. 6]

### 3.3.4 Transição

“O objetivo da fase de transição é substituir o sistema antigo<sup>4</sup> e implementação do novo sistema. Nesta fase, o software é levado a sua comunidade de usuários” [SCH 04, p. 13]. Segundo [RAT 98, p. 6], neste momento geralmente alguns detalhes aparecem e geram a necessidade de novas versões, a correções de alguns problemas ou a finalização de características que foram postergadas.

Entra-se nesta fase quando o produto está pronto para ser utilizado pelos usuário finais, o que requer, de acordo com [RAT 98, p. 6], que um subconjunto usável do sistema esteja pronto com um nível aceitável de qualidade e que a documentação para o usuário esteja pronta. Para se assegurar isso deve-se realizar<sup>5</sup>:

- Testes na versão beta para validar o novo sistema de acordo com as expectativas do usuário.
- Operação paralela com um sistema antigo que está sendo substituído.
- Conversão da base de dados operacional.
- Treinamento dos usuário e mantenedores.
- Apresentação do produto para as equipes de marketing, distribuição e vendas.

Esta fase busca a liberação do sistema para o usuário, para tanto podem ser necessárias diversas iterações gerando versões beta, corrigindo problemas e fazendo aprimoramentos. Para tais atividades é importante estar em contato direto com o usuário final para possibilitar uma assessoria no uso do sistema bem como o feedback sobre o mesmo.

Segundo Schürhaus e Remáculo [SCH 04, p. 13], as atividades fundamentais desta etapa são:

- Finalizar o material de suporte ao usuário final.
- Testar o produto a ser entregue no ambiente do usuário.
- Adequar o produto segundo o feedback do usuário.
- Entregar o produto final ao usuário final.

---

<sup>4</sup>A palavra sistema é aplicada aqui no sentido de forma de realização de um processo, e não apenas no sentido de sistemas computacionais.

<sup>5</sup>Itens retirados de [RAT 98, p. 6]

Os objetivos a serem alcançados aqui são o auto-suporte do usuário, a concordância dos interessados em que o sistema está completo e consistente com os critérios de avaliação da visão do sistema e a liberação da versão final do produto de forma rápida e efetiva.

A dificuldade na realização desta fase é diretamente proporcional à complexidade e ao quão crítico é o sistema. Em [RAT 98, p. 7] faz-se a comparação entre uma nova versão de um sistema para computadores pessoais, cuja fase de transição seria simples, e um sistema nacional de controle de tráfego aéreo, cuja transição seria extremamente complexa.

O quarto e último marco no desenvolvimento é a liberação do produto. Aqui deve ser decidido, segundo [RAT 98, p. 7], se os objetivos foram alcançados, se deve-se iniciar um novo ciclo de desenvolvimento. As principais questões aqui são:

- O usuário está satisfeito?
- Os gastos atuais ainda são aceitáveis frente aos gastos previstos?

### 3.4 Iterações

“Cada fase no Rational Unified Process pode ser quebrada em iterações. Uma *iteração* é um completo ciclo de desenvolvimento resultando em uma versão executável (interna ou externa) do produto, um subconjunto do produto em desenvolvimento, que crescerá de forma incremental de iteração em iteração para se tornar o sistema final” [RAT 98, p. 7, tradução livre].

Alguns benefícios de uma abordagem iterativa são:

- Os riscos são suavizados mais cedo.
- Mudanças são mais controláveis.
- Alto nível de reuso.
- O time de desenvolvimento pode aprender ao longo do caminho.
- Melhor qualidade no geral.

## 3.5 Reengenharia com RUP

De acordo com Kruchten [KRU 03, p. 3], os fundamentos principais de RUP devem ser mantidos mesmo em um projeto de reengenharia:

- suavização antecipada de riscos;
- desenvolvimento iterativo
- avaliação progressiva, baseada em evidências concretas e mensuráveis;
- organização em pequenos times;
- verificação contínua de qualidade;
- gerenciamento de escopo;
- produção apenas dos artefatos necessários.

Kruchten [KRU 03, p. 3] ainda diz que mesmo para projetos de reengenharia deve-se criar documentos como *Visão*, para descrever o que o novo sistema deve atingir; *Planejamento do Projeto*, determinando os marcos a serem alcançados e o que deve ser feito para alcançá-los, as iterações e seus objetivos específicos; uma *Lista de Riscos*, entre outros artefatos.

De início, segundo Kruchten [KRU 03, p. 3-4], deve-se estabelecer um conjunto de artefatos legados do sistema atual (requisitos, arquitetura, testes e documentação de usuário), para que, a partir desses artefatos, possa ser realizado um desenvolvimento normal com RUP. Para se chegar a tais artefatos deve-se passar por uma engenharia reversa<sup>6</sup>.

### 3.5.1 Fluxos de Trabalho

#### 3.5.1.1 Levantamento de Requisitos

Se o sistema legado possuir alguma documentação, esta pode ser adaptada para os formatos usuais em RUP, de forma que devem ser identificados os requisitos antigos organizando-os em casos de uso, conceitos e consultas e também deve-se incluir os novos requisitos.

---

<sup>6</sup>Veja seção 2.2.3, página 31

### 3.5.1.2 Análise

Neste fluxo existem três atividades principais: expansão dos casos de uso e determinação dos eventos do sistema, construção do modelo conceitual, e elaboração dos contratos das operações de sistema.

Caso o sistema legado já tenha uma interface gráfica definida, a expansão dos casos de uso pode ser feita diretamente para os casos de uso reais, aqueles em que se define diretamente a interação do usuário com o sistema, deixando-se de lado os casos de uso essenciais. Isso pode ser justificado citando-se Wazlawick, “a versão essencial é particularmente interessante para que se possam explorar sistemas desconhecidos, sobre os quais o analista ainda precisa aprender para depois poder propor soluções como interfaces” [WAZ 04, p. 64]. Wazlawick ainda diz que “se o sistema for plenamente conhecido ou estiver com sua interface totalmente especificada, pode-se deixar de lado essa versão essencial dos casos de uso e partir diretamente para uma versão real” [WAZ 04, p. 64].

Seria interessante se fazer os casos de uso essenciais caso o analista deseje fazer grandes modificações na interface gráfica do sistema, porém, muitas vezes é indicado que se mantenha a interface compatível com a antiga, seguindo assim o critério de usabilidade da compatibilidade, que pode ser visto em Cybis [dAC 03, p. 42].

O modelo conceitual pode ser construído tendo-se como referência as classes do sistema legado, caso este tenha sido desenvolvido com a tecnologia de orientação a objetos, ou a partir das tabelas do banco de dados, caso seja possível se acessar sua estrutura. Já os contratos podem ter por base as funções executadas a partir dos cliques em botões da interface gráfica legada.

### 3.5.1.3 Projeto

No fluxo de projeto os principais artefatos gerados são os diagramas de interação baseados nos contratos e o diagrama de classes.

O diagrama de interação é feito a partir dos contratos da análise, e pode ser elaborado contando-se com as funções do sistema legado para melhorar a compreensão do sistema.

O diagrama de classes pode ser construído da mesma forma num processo de engenharia normal e em um processo de reengenharia já que seu conteúdo é baseado totalmente em artefatos do próprio RUP.



### 3.5.1.4 Implantação

Segundo Kruchten [KRU 03, p. 5], a implantação de um sistema após uma reengenharia pode ser mais complicada do que a de um sistema totalmente novo. Pode-se simplesmente retirar o sistema antigo e implantar o novo ou deixar os dois trabalhando em paralelo até que se tenha confiança o bastante no novo sistema para que este possa ser usado sozinho.

Para Kruchten [KRU 03, p. 5], a implantação de um sistema fruto de uma reengenharia traz problemas que não existem em um sistema novo como conversão de dados, continuidade de operações e re-treinamento de pessoal.

### 3.5.2 Fases

Na fase de *concepção*, segundo Kruchten [KRU 03, p. 5], deve-se desenvolver, além dos documentos normais como visão, um documento especificando os artefatos que terão que ser reconstruídos, além de se iniciar o processo de engenharia reversa para alguns artefatos, principalmente de requisitos e análise.

Para Kruchten [KRU 03, p. 5], na fase de *elaboração*, os artefatos vindos do sistema antigo devem ser todos adaptados, atualizados ou reconstruídos para que o desenvolvimento possa seguir quase como um desenvolvimento RUP normal.

“A fase de *construção* não tem nenhuma diferença significativa em relação a qualquer outro projeto RUP. Elementos adicionais passar por uma engenharia reversa, são re-projetados e documentados da forma necessária. Ou são apenas traduzidos para uma nova linguagem” [KRU 03, p. 5].

Na fase de *transição* encontram-se elementos mais delicados. Aqui é dada maior ênfase ao fluxo de implantação<sup>7</sup>, cujo desafio é determinar a estratégia utilizada para a migração do sistema antigo para o novo.

---

<sup>7</sup>Veja a seção 3.5.1.4, na página 56

## 4 *Estudo de Caso: APUFSC*

O sistema da APUFSC é um software de controle da associação que engloba controle do cheque APUFSC, de planos de saúde dos associados e dependentes e de contabilidade da associação. Para tanto, o sistema conta também com alguns cadastros como de professores, convênios, bancos, planos de saúde, dependentes dos associados, etc.

O atual sistema da APUFSC foi implementado com o Microsoft Access® a partir de pouca documentação. O Microsoft Access® é um sistema gerenciador de bancos de dados com “ferramentas que são sofisticadas o suficiente para desenvolvedores profissionais, além disso, são de fácil aprendizagem para novos usuários” [MIC 03].

A reengenharia desse sistema é motivada principalmente por duas questões, a melhoria nas condições de manutenção do sistema, atualmente precárias pela falta de documentação, e a desvinculação do sistema de uma plataforma proprietária como é o caso do Microsoft Access®.

### 4.1 Reengenharia do Sistema da APUFSC

Neste estudo de caso foi realizada uma concepção inicial do sistema<sup>1</sup> seguida da execução dos fluxos<sup>2</sup> de requisitos, análise, projeto e implementação do RUP. A concepção foi realizada considerando-se o sistema como um todo, já os fluxos foram realizados apenas para um caso de uso.

O processo de reengenharia do sistema da APUFSC começou com uma entrevista com o desenvolvedor do sistema atual. Tal entrevista visava a obtenção de conhecimento sobre o sistema, sua implementação e de como tem sido usado na APUFSC.

A partir de tais informações foi possível dar início ao processo de engenharia reversa a fim de se gerar a documentação mínima necessária para se realizar a engenharia pro-

---

<sup>1</sup>Veja seção 3.3.1, página 48.

<sup>2</sup>Veja seção 3.1, página 36.

gressiva, ou seja, o processo de desenvolvimento do novo sistema da APUFSC.

A engenharia reversa teve seu início com uma análise da estrutura do sistema, descrita a seguir. A descrição do restante do processo é feita aqui em termos de fluxos de trabalho. O levantamento de requisitos foi feito para todo o sistema, enquanto que os fluxos de análise, projeto e implementação da camada de domínio foram realizados apenas para um caso de uso.

A abordagem de apenas um caso de uso deve-se ao desejo de se avaliar o impacto das adaptações feitas no RUP para o processo de reengenharia até um estágio avançado do desenvolvimento, o que seria difícil de se fazer para todo o sistema no pouco tempo em que este trabalho foi desenvolvido.

#### 4.1.1 Estrutura do atual sistema

O atual sistema da APUFSC é um banco de dados formado por quarenta e uma tabelas. Dessas, grande parte são tabelas que abstraem alguma entidade do mundo real, porém, outras são usadas apenas para controle interno. A tabela 2 mostra uma lista de algumas tabelas do banco de dados do atual sistema divididas em abstrações e tabelas de uso interno do sistema.

<b>Abstrações</b>	<b>Uso Interno</b>
Agencia	Aux Despesas PS por Mes
Associado a Plano de Saúde	Aux para Relatorio de Contas
Banco	Aux Relatorio A-D
Centro	Aux utilizacao / categoria
Cheque APUFSC	BB - valores padrão
Códigos de Dependência	Besc - valores padrão
Consolidação de despesas	Besc - detalhe tab
Contas	Caixa - valores padrão
Contato	Data
Convênio	Meses

Tabela 2: Listagem de algumas tabelas do sistema antigo da APUFSC.

Além das tabelas, o sistema também contém um grande número de consultas. Cada consulta é uma interação do restante do sistema com as tabelas. Uma consulta pode ser de um dos seguintes tipos:

- **Consulta seleção.** Recupera dados de uma ou mais tabelas usando critérios específicos em uma ordem pré-definida. Algumas consultas seleção usadas no sistema APUFSC são: Banco onde APUFSC tem conta, Centro-Departamento e Professor / Departamento.
- **Consulta acréscimo.** Adiciona um grupo de registros ao final de uma ou mais tabelas. Exemplo: Cria mensalidades zeradas.
- **Consulta exclusão.** Exclui um grupo de registros de uma ou mais tabelas. Exemplo: Apaga consolidação geral.
- **Consulta atualização.** Faz alterações globais em um grupo de registros em uma ou mais tabelas. Exemplo: I - Atualiza nomes de agencias.
- **Consulta união.** Utiliza o operador UNIÃO do Access para combinar os resultados de duas ou mais consultas seleção. É escrito em SQL. Exemplos: Caixa - todos os registros, BB - todos os registros.
- **Consulta de tabela de referência cruzada.** Calcula e reestrutura dados a fim de realizar uma análise mais fácil deles. Exemplo: C Despesas por tipo.

Os formulários definidos no Access são a interface gráfica utilizada pelos usuários para interagir com o sistema. O sistema da APUFSC possui sessenta e cinco formulários que fazem a ponte entre o usuário e as consultas e tabelas já descritas e também dão acesso aos relatórios que serão descritos a seguir. Para cada funcionalidade do sistema existe ao menos um formulário.

Os relatórios são estruturas montadas para exibir os dados de uma forma específica. No atual sistema da APUFSC existem quarenta e três diferentes relatórios, mostrando desde listagem de professores por centro até extratos de cheques APUFSC de um professor em um mês.

Além das estruturas já descritas, o sistema ainda conta com algumas macros e módulos. Macros são conjuntos de ações na qual cada uma realiza uma operação específica, como abrir um formulário ou um relatório. Já um módulo é um conjunto de declarações, instruções e procedimentos escritos em Microsoft Visual Basic que são armazenados conjuntamente formando uma unidade.

## 4.1.2 Levantamento de Requisitos

Da entrevista feita com o desenvolvedor do atual sistema, foi possível determinar dois principais requisitos que justificam a reengenharia. O primeiro é o desejo de se desvincular o sistema de uma plataforma proprietária, o que poderia diminuir os custos de uso do sistema em outras máquinas, e mesmo de adaptação e implantação do sistema em outras associações semelhantes. O segundo é a necessidade de melhor documentação a fim de se facilitar os esforços em manutenção e também possíveis adaptações do sistema para outras associações congêneres.

A listagem dos requisitos suplementares identificados pode ser encontrada na seção A.1 do Apêndice.

Além disso, a partir da entrevista e de uma posterior análise mais aprofundada do sistema, foram identificados os principais processos que deveriam ser atendidos pelo novo sistema, ou seja, aqueles que deveriam ser descritos em casos de uso.

Na análise do sistema atual, os itens que mais auxiliaram na identificação dos casos de uso foram os formulários, já que estes mostram os pontos de interação do usuário com o sistema. A partir da análise da tela inicial (figura 5) já é possível se observar os grandes grupos de funcionalidades do sistema: Cheques APUFSC, Contabilidade e Planos de Saúde. A tela inicial também possui um grupo de cadastros, porém, estes poderão ser considerados como operações de manutenção de conceitos<sup>3</sup>. Assim, analisando-se os itens presentes em cada grupo e os formulários referentes a cada um, foi possível determinar quais realmente representam alguma funcionalidade importante e complexa o bastante para justificar um caso de uso.

Dentre as opções presentes em cada um dos grupos verificou-se que os itens *Edição de Cheques*, *Estornos*, *Segurados/Dependentes*, *Digitação de Despesas* e *Edição de Despesas* são meras operações de manutenção de alguns dos conceitos do sistema. Enquanto os itens *Entrega de Talão*, *Digitação de Cheques*, *Contas*, *Mensalidades*, *Relatório de Atualizações* e *Consolidação de despesas mensais* poderiam ser mapeados, respectivamente, para os casos de uso *Entregar talão de cheques*, *Digitar cheque*, *Lançamentos*, *Inserir ano de mensalidades de plano de saúde*, *Gerar relatório de atualizações em plano de saúde* e *Gerar relatório para banco*.

Além dos casos de uso já citados, também foi identificado o caso de uso *Solicitar atualização em plano de saúde*, que no atual sistema da APUFSC é uma funcionalidade

---

<sup>3</sup>Veja a tabela 5, página 69

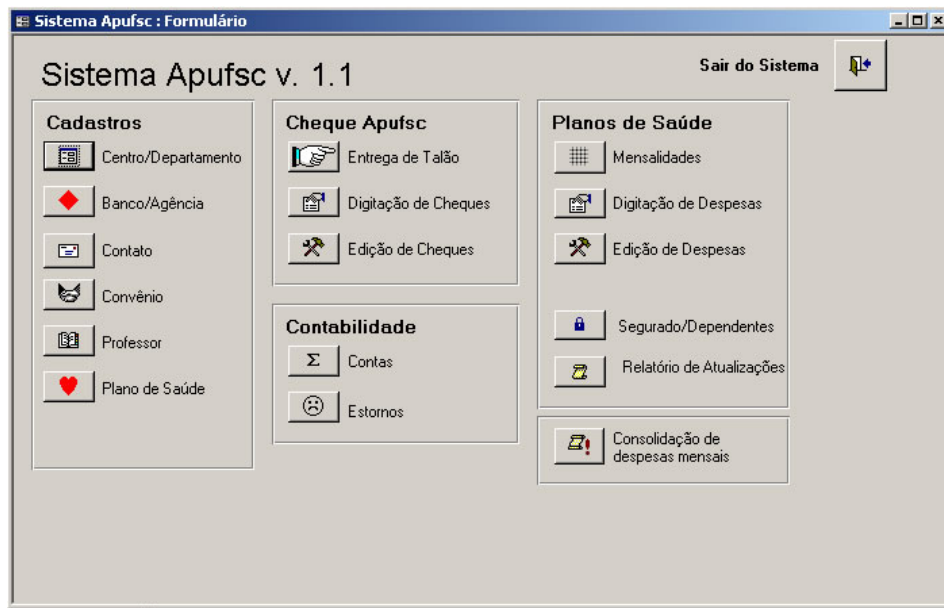


Figura 5: Tela inicial do atual sistema da APUFSC.

da tela de cadastro de Segurados/Dependentes, porém entendeu-se que, por ser uma funcionalidade que não faz parte da simples operação de manutenção de associados a planos de saúde, seria interessante separar a solicitação de atualização em planos de saúde.

Uma descrição um pouco mais detalhada dos casos de uso pode ser encontrada na tabela 4, página 68 do Apêndice.

Ainda como parte do processo de levantamento de requisitos, foram identificados os principais conceitos do sistema. Tal listagem foi montada a partir das tabelas do sistema atual da APUFSC, deixando-se apenas aquelas que tinham algum significado no mundo real e descartando-se, neste momento, todas as tabelas que eram usadas apenas para algum tipo de controle interno do sistema. A listagem dos conceitos pode ser vista na tabela 5, página 69. Nesta tabela, as colunas I, A, E e C indicam, respectivamente, se o conceito deve sofrer operações de Inserção, Alteração, Exclusão e Consulta.

Por fim, a partir dos relatórios do sistema atual da APUFSC foi montada uma listagem das consultas a serem realizadas pelo novo sistema. Tal listagem mostra os conceitos que devem ser consultados e as possíveis variações na forma de consulta, por exemplo, além de se consultar os departamentos da universidade, pode-se agrupá-los por centros ou filtrá-los listando apenas os departamentos de um determinado centro. A listagem completa das consultas identificadas se encontra na seção A.1.4, página 69, do Apêndice.

### 4.1.3 Análise

O primeiro passo na análise deve ser a expansão dos casos de uso. Tal atividade deve ser feita para todos os casos de uso listados no levantamento de requisitos. Esta expansão deve ser feita diretamente para casos de uso reais uma vez que neste ponto já se tem um bom conhecimento dos formulários do atual sistema, o que facilita na identificação dos passos do fluxo principal, na definição da interface e do diagrama de sequência.

Para cada caso de uso devem ser determinadas as seguintes partes:

- **Nome e descrição.** Itens já determinados na listagem de casos de uso feita no levantamento de requisitos<sup>4</sup>.
- **Atores.** Lista de todos os que interagem com o sistema durante o caso de uso. Não aparece em todos os casos de uso.
- **Interessados.** Lista de todas as entidades interessadas no resultado do caso de uso. Não aparece em todos os casos de uso.
- **Pré-condições.** Situações que devem ser verdadeiras antes de se iniciar o caso de uso. Não são testadas durante o caso de uso e caso sejam falsas inviabilizam a execução do mesmo.
- **Fluxo principal.** Conjunto de passos que definem a ocorrência de sucesso do processo descrito pelo caso de uso. Os passos marcados com [EV] são eventos disparados pelo usuário, aqueles marcados com [RS] são as respostas do sistema. Este fluxo pode ser construído tendo-se como base o fluxo da funcionalidade correlata no atual sistema.
- **Tratamento de exceções.** Descrição de tudo o que pode acontecer de errado em cada passo do fluxo principal e determinação dos fluxos alternativos para cada exceção. A identificação das exceções pode ser feita também pelo sistema atual buscando-se nele as situações já previstas.
- **Interfaces gráficas relacionadas.** Esboços de todas as telas envolvidas diretamente na realização do caso de uso. Podem ser feitas tendo-se como base os formulários do sistema atual.

---

<sup>4</sup>Veja a tabela 4, na página 68.

- **Diagrama de sequência.** Um diagrama que mostra, passo a passo, toda a sequência de mensagens trocadas entre usuário e interface gráfica e interface gráfica e sistema. A construção deste diagrama pode ser facilitada tendo-se como base as trocas de mensagens entre os formulários e as consultas e tabelas do atual sistema.

Neste estudo de caso, apenas um caso de uso foi expandido, o resultado desta atividade está na seção A.2 do Apêndice.

O passo seguinte foi a construção do modelo conceitual, partindo-se da listagem de conceitos realizada no levantamento de requisitos<sup>5</sup> e das tabelas do atual sistema da APUFSC. O modelo conceitual está dividido entre as figuras 9 e 10, na seção A.3, por ser grande demais para ser apresentado em apenas uma imagem. A identificação dos atributos de cada conceito foi feita a partir das colunas das tabelas do atual banco de dados, enquanto as associações foram identificadas através das colunas das tabelas que eram usadas como chave estrangeira, ou seja, que referenciavam uma chave de outra tabela. O modelo conceitual também possui algumas associações temporárias, aquelas marcadas com o estereótipo << *temp* >>. Tais associações foram identificadas durante a construção do diagrama de sequência (figura 8, página 73) devido aos laços de repetição, que exigem o uso constante de informações como o Banco selecionado.

Por fim, o fluxo de análise termina com a elaboração dos contratos, um para cada consulta identificada no levantamento de requisitos e um para cada evento identificado nos casos de uso expandidos.

Para este estudo de caso foram elaborados apenas os contratos referentes ao caso de uso *Lançar receitas e despesas*, sendo que os eventos e consultas foram identificados a partir do diagrama de sequência deste caso de uso<sup>6</sup>. Os contratos elaborados podem ser encontrados na seção A.4 do Apêndice. As palavras *grifadas* nos contratos representam parâmetros passados pelo evento ou conceitos e atributos do modelo conceitual.

#### 4.1.4 Projeto

O fluxo de projeto tem duas atividades principais, a confecção dos diagramas de colaboração e do diagrama de classes. Para cada contrato elaborado durante a análise é feito um diagrama de colaboração a fim de se identificar as mensagens passadas entre os objetos envolvidos na execução do contrato. Na seção A.5 do Apêndice estão os diagramas

---

<sup>5</sup>Veja a tabela 5, página 69.

<sup>6</sup>Veja a figura 8, página 73.



de colaboração referentes aos contratos exibidos na seção A.4.

Feitos os contratos, o passo seguinte é a confecção do diagrama de classes de projeto. Esse diagrama se baseia no modelo conceitual, com a adição de algumas informações como o sentido das associações e os métodos de cada classe. O sentido das associações é definido a partir do sentido em que as mensagens trafegam nos diagramas de colaboração, e os métodos são encontrados a partir das próprias mensagens. É importante notar que nem todas as mensagens existentes nos diagramas de colaboração aparecem no diagrama de classes. Mensagens básicas de criação de instâncias, consulta a atributos e criação de associações podem ser deixadas de fora do diagrama de classes. O diagrama de classes resultante dos contratos do caso de uso estudado está na seção A.6 do Apêndice.

#### 4.1.5 Implementação

O último passo feito no estudo de caso aqui descrito, foi a implementação da camada de domínio. Tal atividade foi feita a partir dos artefatos gerados no projeto, ou seja, diagramas de colaboração e diagrama de classes do projeto. O código foi gerado na linguagem Java.

Para cada classe do diagrama de classes do projeto, foi criada uma classe na implementação, e dos atributos das classes do diagrama surgiram variáveis de instância privadas de tipo básico. Para cada variável desse tipo também foram criados métodos de acesso, os get para leitura e set para escrita.

As associações do diagrama de classes também foram representados no código como variáveis de instância privadas. Como as associações do diagrama de classes deste projeto são todas direcionadas, basta que a classe origem da associação tenha uma variável de instância do tipo da classe destino da associação.

Neste projeto, nem todas as variáveis resultantes de associações tiveram todos os métodos de acesso implementados. Os métodos de interação com tais variáveis foram determinados a partir das mensagens dos diagramas de colaboração.

O código gerado nesta fase do desenvolvimento do estudo de caso pode ser visto na seção A.7 do Apêndice.

## 5 *Conclusão*

Um processo de reengenharia pode ser visto sob dois diferentes prismas, a reengenharia de negócios e a reengenharia de sistemas. A primeira, também chamada de BPR, é muito mais abrangente, envolve toda uma organização e acaba, em última instância, levando também à segunda.

A reengenharia de sistemas, foco deste trabalho, pode ser vista como uma atividade composta, basicamente, dos seguintes passos: conhecer o software existente e as razões que levam à necessidade de uma reengenharia; fazer a engenharia reversa do sistema existente a fim de se obter conhecimento e documentação o bastante sobre ele para se realizar a construção de um novo sistema; e, realizar a construção do novo sistema partindo-se da documentação gerada com a engenharia reversa e dos requisitos que levaram ao processo de reengenharia.

Neste trabalho verificou-se o poder de adaptação do Rational Unified Process quando este pôde ser perfeitamente utilizado em um processo de reengenharia. Num primeiro momento, usou-se a fase de concepção do RUP para se estudar o sistema existente e levantar os requisitos que motivaram a realização da reengenharia.

Em seguida, foi possível se utilizar, com algumas pequenas adaptações, os fluxos de levantamento de requisitos, análise e projeto de RUP para se estudar e documentar o sistema existente a fim de preparar a base para a construção do novo sistema. Dentre as adaptações feitas pode-se citar:

- identificação de requisitos a partir da análise do sistema já existente;
- identificação de conceitos a partir da estrutura de dados ou banco de dados do sistema legado;
- identificação dos casos de uso e consultas a partir da estrutura e dos processos dos sistema;

- expansão dos casos de uso diretamente para casos de uso reais utilizando-se como modelo de interface gráfica a interface do próprio sistema existente;
- elaboração dos contratos usado-se como base, além dos casos de uso, as funções do sistema legado.

A construção do sistema a partir dos artefatos gerados com o auxílio de uma engenharia reversa acaba sendo praticamente igual à construção de um sistema qualquer. Um dos grandes desafios de uma reengenharia surge no final, na implantação do sistema, quando deve-se tomar cuidado com a estratégia utilizada afim de se evitar problemas com os usuários finais.

O processo de reengenharia do sistema da APUFSC não foi completado principalmente por dois motivos: o pouco tempo de desenvolvimento deste trabalho e a falta de mão-de-obra para o desenvolvimento. Porém, mesmo não se tendo feito o processo todo, foi possível avaliar bem o uso do RUP em um projeto dessa natureza e algumas adaptações interessantes foram propostas.

# *APÊNDICE A – Documentação*

## **A.1 Levantamento de Requisitos**

### **A.1.1 Requisitos suplementares**

A tabela a seguir mostra a listagem dos requisitos suplementares identificados durante o processo de reengenharia do sistema da APUFSC<sup>1</sup>.

<b>Nome</b>	<b>Descrição</b>
Implementação	A implementação do sistema deve ser feita usando-se ferramentas livres e a linguagem usada deve ser Java.
Documentação	O sistema deve ser bem documentado para facilitar futura manutenção.
Cancelamento	A qualquer instante, um cadastro ou edição em andamento pode ser cancelado.

Tabela 3: Requisitos suplementares para o processo de reengenharia.

### **A.1.2 Listagem de casos de uso**

<b>Nome</b>	<b>Descrição</b>
Lançar receitas e despesas	Lançamento detalhado de receitas e despesas da associação.
Solicitar atualização em plano de saúde	Solicitação por parte de um associado a um plano de saúde para inclusão, exclusão (definitiva ou temporária), alteração, solicitação de segunda via ou transferência de plano.
Gerar relatório de atualizações de plano de saúde	Geração de relatórios com as atualizações cadastrais dos associados com os planos e consolidação das atualizações no banco de dados.
Gerar relatório para banco	Consolidação das contas bancárias dos associados e geração de relatórios e/ou arquivo de débito automático para os bancos.
Entregar talão de cheques	Entrega de um talão de cheques APUFSC a um conveniado.

<sup>1</sup>Veja a seção 4.1.2, página 60.

Digitar cheque	Digitação dos dados de um cheque passado por um associado a um conveniado.
Inserir ano de mensalidades de plano de saúde	Inserção dos valores de mensalidade para um plano e seus planos complementares em todos os meses de um novo ano.

Tabela 4: Listagem de casos de uso identificados no levantamento de requisitos.

### A.1.3 Listagem dos conceitos

Nesta tabela, as colunas I, A, E e C correspondem, respectivamente, às operações de Inserção, Alteração, Exclusão e Consulta aos conceitos listados.

Conceito	I	A	E	C	Observações
Professor	X	X		X	O sistema atual não permite exclusão.
Centro	X	X	X	X	O sistema atual não permite exclusão, porém pode ser excluído se não houver professor algum associado.
Departamento	X	X	X	X	O sistema atual não permite exclusão, porém pode ser excluído se não houver professor algum associado.
Contato	X	X		X	O sistema atual não permite exclusão. Pode-se apenas indicar exclusão em campo booleano. Associado a Tipo de Convênio.
Convênio	X	X		X	O sistema atual não permite exclusão. Pode-se apenas indicar exclusão em campo booleano. Associado a Tipo de Convênio.
Banco	X	X	X	X	O sistema atual não permite exclusão, porém pode ser excluído caso não tenha contatos e/ou lançamentos associados.
Agência	X	X	X	X	O sistema atual não permite exclusão, porém pode ser excluído caso não tenha contatos e/ou lançamentos associados.
Conta Bancária	X	X	X	X	Pode ser excluído caso não tenha contatos e/ou lançamentos associados.
Lançamento	X	X		X	O sistema atual não permite exclusão. Débitos/créditos em banco.
Consolidação		X		X	
Despesa Consolidada		X	X	X	
Cheque APUFSC	X	X		X	O sistema atual não permite exclusão.
Estorno	X	X		X	O sistema atual não permite exclusão.
Planos de Saúde	X	X	X	X	O sistema atual não permite exclusão, porém pode-se excluir se não houver professor e despesa alguma associada.

Planos Complementares	X	X	X	X	O sistema atual não permite exclusão, porém pode-se excluir se não houver professor e despesa alguma associada.
Modalidade do Plano de Saúde	X	X	X	X	O sistema atual não permite exclusão, porém pode-se excluir se não houver associado algum nem mensalidades cadastradas.
Associado a Plano de	X	X	X	X	Operações devem ser confirmadas pela operadora do Plano de Saúde.
Mensalidade de Plano de Saúde	X	X		X	
Operação Plano	X	X		X	
Dependente em Plano de Saúde	X	X	X	X	Depende de associado. Operações devem ser confirmadas pela operadora do Plano de Saúde.
Grau de Dependência	X	X		X	

Tabela 5: Listagem dos conceitos identificados no levantamento de requisitos.

#### A.1.4 Listagem das consultas

- Departamentos
  - Agrupados por centro
  - Filtrados por centro
- Centros
- Agências
  - Agrupadas por banco
  - Filtradas por centro
- Bancos
  - Em que a APUFSC tem conta
  - Em que os associados têm conta
- Contatos
  - Nomes e endereços
  - Agrupados por categorias

- Filtradas por categorias
- Convênios
- Professores
  - Nomes e telefones
  - Nomes e matrículas
  - Dados bancários agrupados por banco
- Cheques APUFSC
  - Agrupados por convênio e filtrados por período
- Receitas e despesas da APUFSC
  - Agrupadas por banco e filtradas por período
  - Agrupadas por tipo e filtradas por período
  - Receitas filtradas por período
- Estornos
  - Quitados
- Associados a planos de saúde
  - Agrupados por plano de saúde
  - Filtrados por plano de saúde
- Operações de planos de saúde
- Despesas
  - Agrupados por plano de saúde e filtrados por período
  - Filtrados por plano de saúde e por período
  - Por associado e filtradas por período
  - Consolidadas filtradas por período

## A.2 Casos de Uso Expandidos

### Lançar Receitas e Despesas

Lançamento detalhado de receitas e despesas da associação.

#### Fluxo Principal:

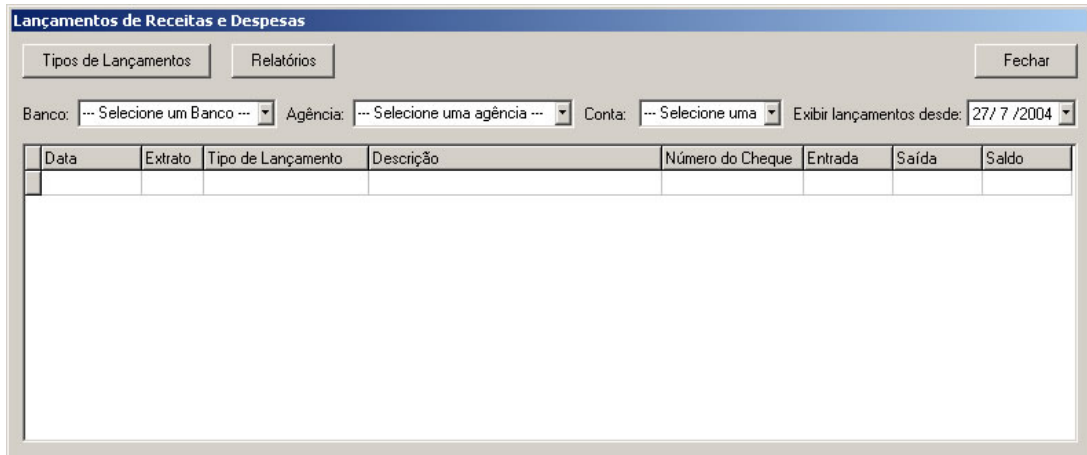
1. O usuário acessa a tela de lançamentos de receitas e despesas.
2. O usuário seleciona o banco para o qual deseja registrar os lançamentos.
3. O usuário seleciona a agência para a qual deseja registrar os lançamentos.
4. O usuário seleciona a conta para o qual deseja registrar os lançamentos.
5. [EV] Para cada lançamento para a conta selecionada, o usuário preenche, na linha habilitada da tabela, os campos “Data”, “Extrato”, “Tipo de Lançamento”, “Descrição”, “Número do Cheque” e o valor de “Entrada” ou “Saída” pressionando TAB ao final do preenchimento.
6. O usuário clica no botão “Fechar”.

#### Tratamento de Exceções:

- 5a O tipo de lançamento não está cadastrado.
  - 5a.1. O usuário clica no botão “Tipos de Lançamentos”.
  - 5a.2. O sistema abre a tela “Tipos de Lançamentos”.
  - 5a.3. O usuário clica no botão “Adicionar”
  - 5a.4. [EV] O usuário entra com o nome do novo tipo de lançamento no campo “Descrição” habilitado.
  - 5a.5. O usuário clica no botão “Fechar”.
  - 5a.6. O sistema volta para a tela “Lançamento de Receitas e Despesas”.
  - 5a.7. Retorna ao fluxo principal no passo 4.

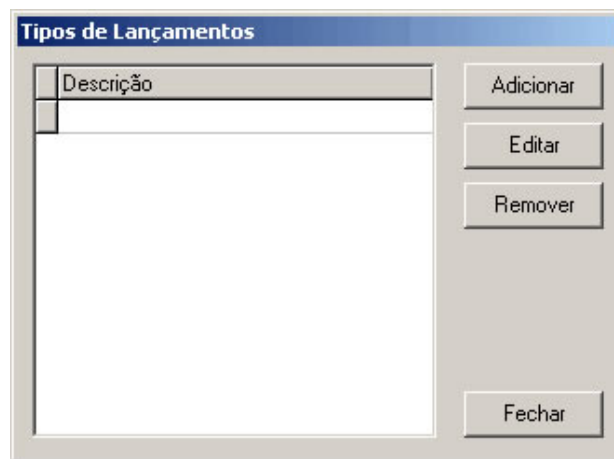


## Interface gráfica:



The screenshot shows a window titled "Lançamentos de Receitas e Despesas". At the top, there are two tabs: "Tipos de Lançamentos" and "Relatórios", and a "Fechar" button on the right. Below the tabs, there are three dropdown menus for "Banco: ... Selecione um Banco ...", "Agência: ... Selecione uma agência ...", and "Conta: ... Selecione uma ...". To the right of these is a date field "Exibir lançamentos desde: 27/7/2004". Below the filters is a table with the following columns: "Data", "Extrato", "Tipo de Lançamento", "Descrição", "Número do Cheque", "Entrada", "Saída", and "Saldo". The table is currently empty.

Figura 6: Interface principal do caso de uso Lançar Receitas e Despesas.



The screenshot shows a window titled "Tipos de Lançamentos". It features a list box with a single entry labeled "Descrição". To the right of the list box are three buttons: "Adicionar", "Editar", and "Remover". At the bottom right of the window is a "Fechar" button.

Figura 7: Interface de suporte do caso de uso Lançar Receitas e Despesas.

### Diagrama de seqüência:

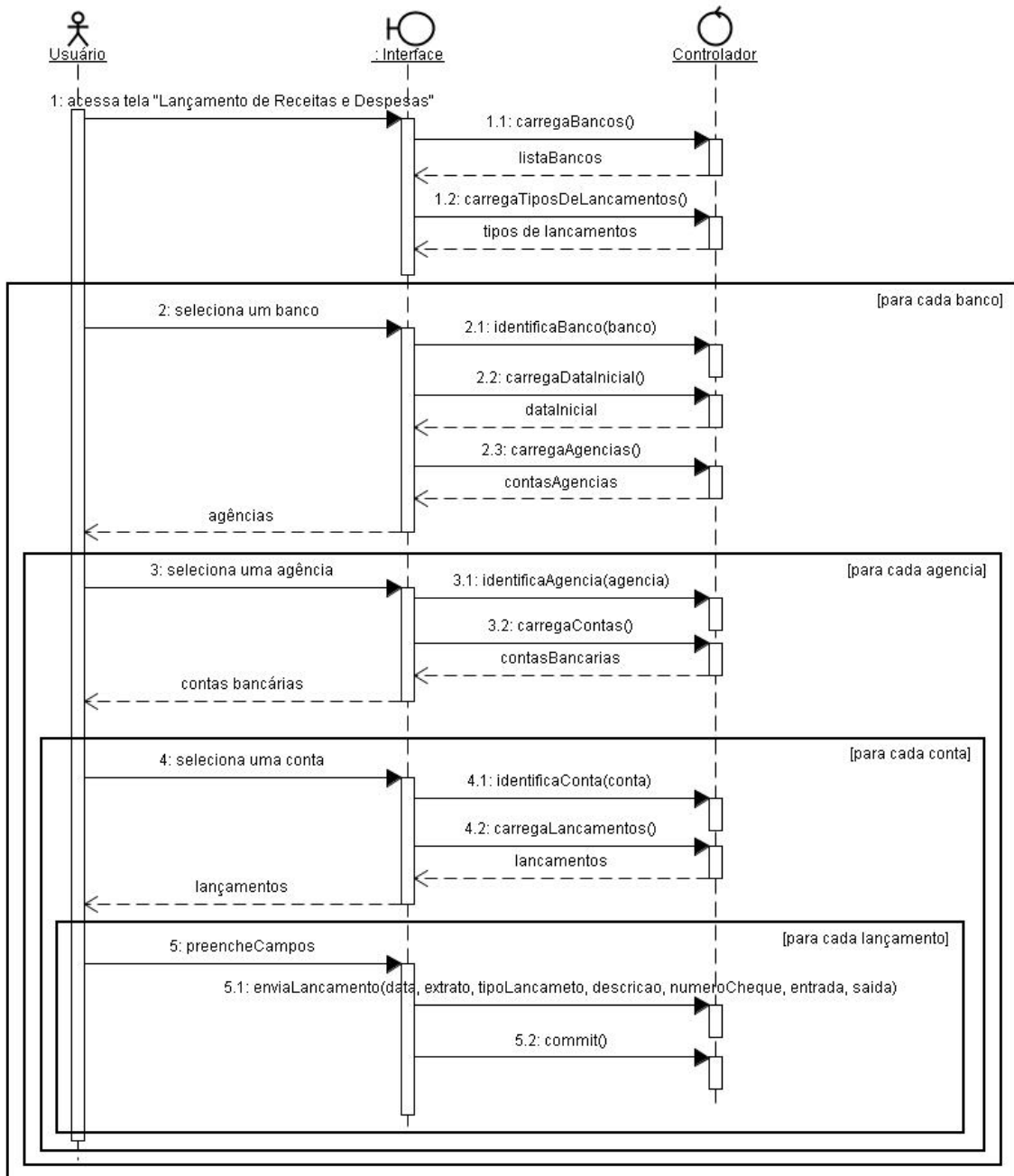


Figura 8: Diagrama de seqüência do caso de uso Lançar Receitas e Despesas.

## A.3 Modelo Conceitual

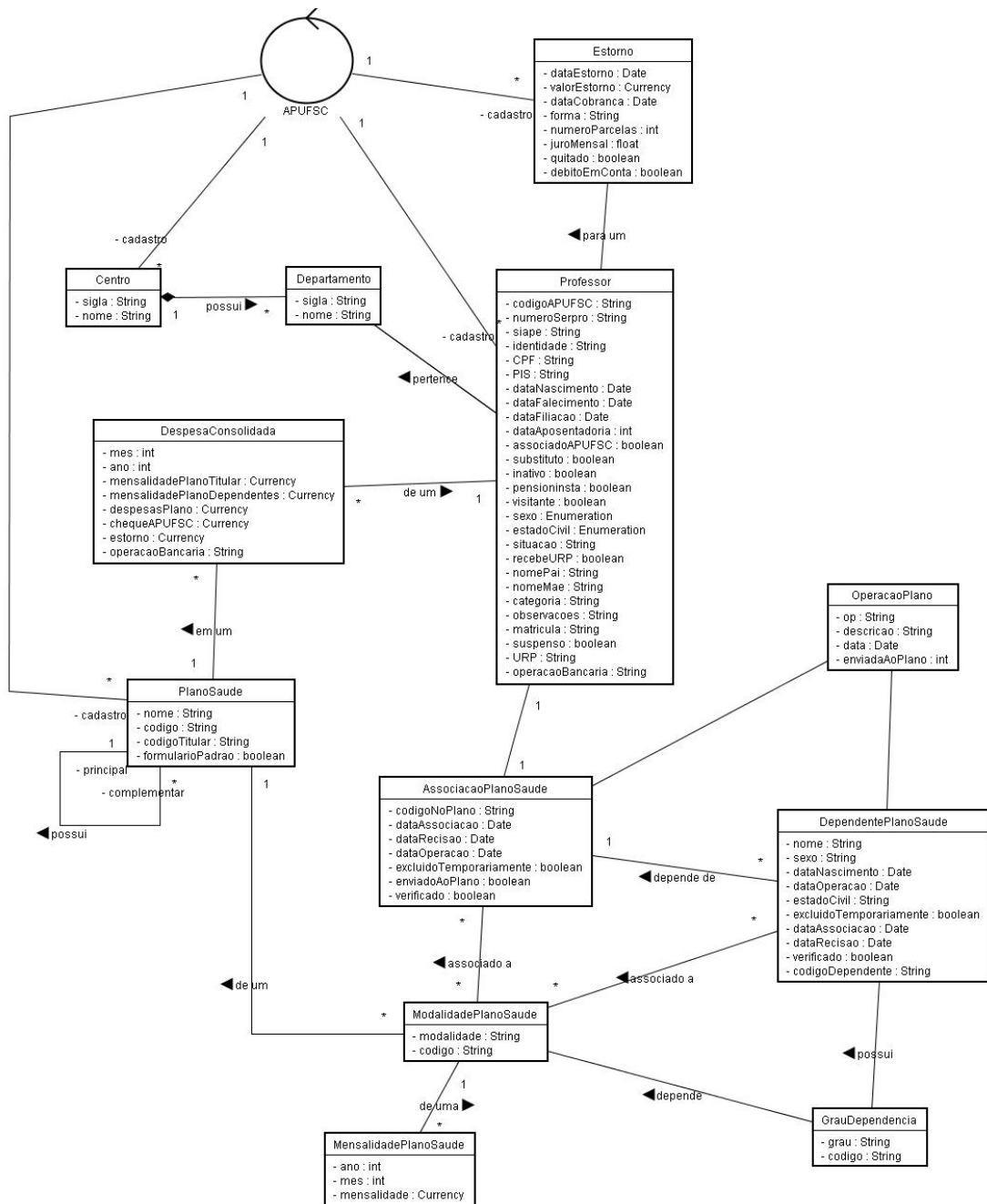


Figura 9: Primeira parte do modelo conceitual do sistema.

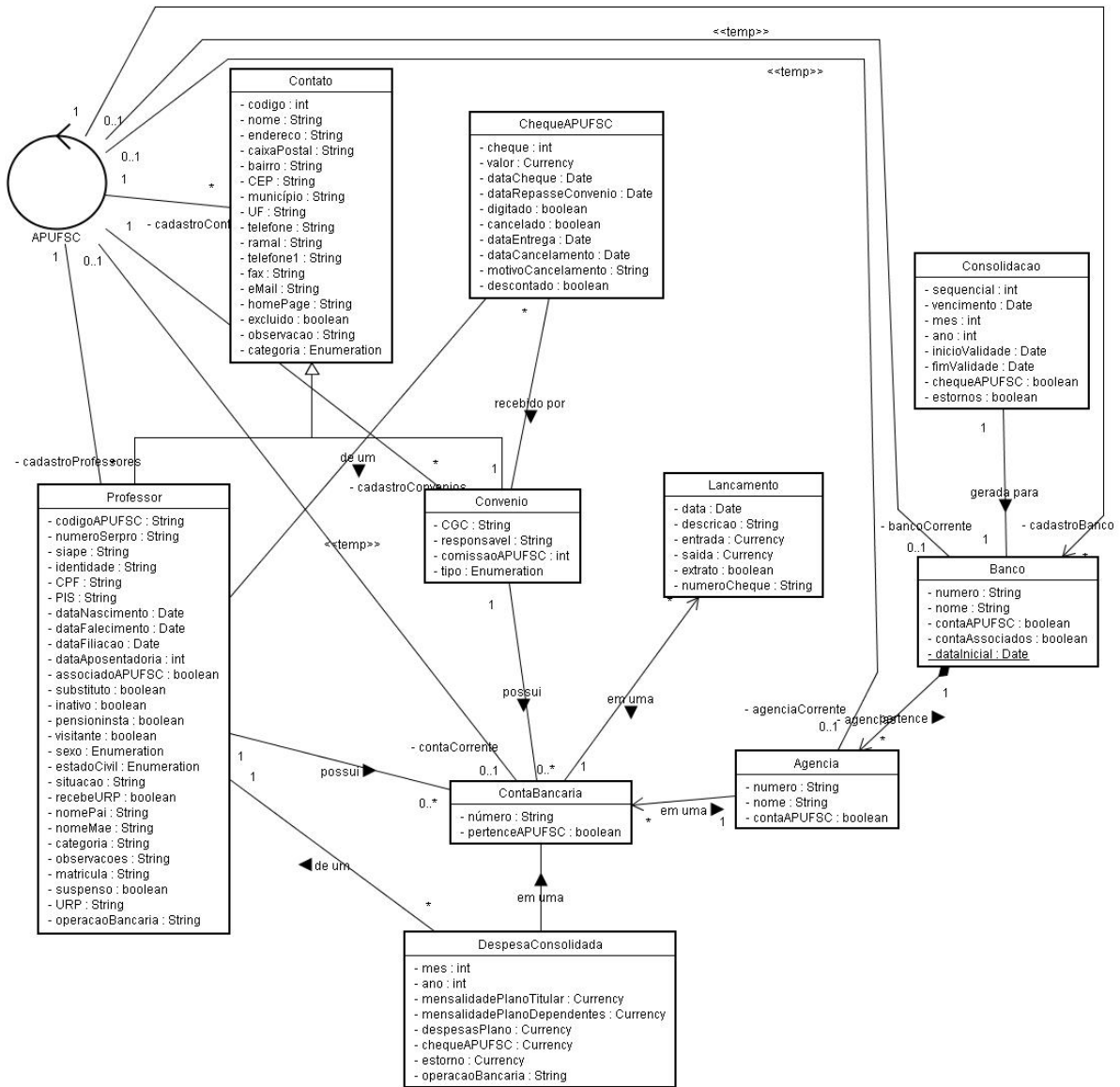


Figura 10: Segunda parte do modelo conceitual do sistema.

## A.4 Contratos

Aqui são listados os contratos referentes ao caso de uso exibido na seção A.2.

<b>Nome:</b>	carregaBancos()
<b>Responsabilidades:</b>	Carregar a lista de bancos cadastrados no sistema.
<b>Pré-condições:</b>	
<b>Resultados:</b>	Retorna o <i>nome</i> de todos os <i>Banco</i> cadastrados que possuem <i>contaAPUFSC=true</i> .

<b>Nome:</b>	carregaTiposDeLancamentos()
<b>Responsabilidades:</b>	Carregar a lista de tipos de lançamentos disponíveis para o usuário.
<b>Pré-condições:</b>	
<b>Resultados:</b>	Retorna o todos os <i>TiposDeLancamento</i> cadastrados.

<b>Nome:</b>	identificaBanco(banco: String)
<b>Responsabilidades:</b>	Identificar o banco selecionado pelo usuário.
<b>Pré-condições:</b>	Existe um <i>Banco</i> com <i>nome=banco</i> .
<b>Pós-condições:</b>	O <i>Banco</i> foi associado como <i>bancoCorrente</i> .
<b>Exceções:</b>	

<b>Nome:</b>	carregaDataInicial()
<b>Responsabilidades:</b>	Carregar a data inicial dos registros do banco corrente.
<b>Pré-condições:</b>	Existe um <i>bancoCorrente</i> .
<b>Resultados:</b>	Retorna a <i>dataInicial</i> do <i>bancoCorrente</i> .

<b>Nome:</b>	carregaAgencias()
<b>Responsabilidades:</b>	Carregar a lista de agências do banco corrente.
<b>Pré-condições:</b>	Existe um <i>bancoCorrente</i> .
<b>Resultados:</b>	Retorna o <i>nome</i> de todas as <i>Agencia</i> em que <i>contaAPUFSC=true</i> e que estão associadas ao <i>bancoCorrente</i> .

<b>Nome:</b>	identificaAgencia(agencia: String)
<b>Responsabilidades:</b>	Identificar a agência selecionada pelo usuário.
<b>Pré-condições:</b>	Existe uma <i>Agencia</i> com <i>nome=agencia</i> . Existe um <i>bancoCorrente</i> . A <i>agencia</i> está associada ao <i>bancoCorrente</i> .
<b>Pós-condições:</b>	A <i>agencia</i> foi associado como <i>agenciaCorrente</i> .
<b>Exceções:</b>	

<b>Nome:</b>	carregaContas()
<b>Responsabilidades:</b>	Carregar a lista de contas bancárias da agência corrente.
<b>Pré-condições:</b>	Existe uma <i>agenciaCorrente</i> .
<b>Resultados:</b>	Retorna o <i>numero</i> de todas as <i>ContaBancaria</i> em que <i>pertenceAPUFSC=true</i> e que estão associadas à <i>agenciaCorrente</i> .

<b>Nome:</b>	identificaConta(conta: String)
<b>Responsabilidades:</b>	Identificar a conta bancária selecionada pelo usuário.
<b>Pré-condições:</b>	Existe uma <i>ContaBancaria</i> com <i>numero=conta</i> . Existe uma <i>agenciaCorrente</i> . A <i>conta</i> está associada à <i>agenciaCorrente</i> .
<b>Pós-condições:</b>	A <i>conta</i> foi associada como <i>contaCorrente</i> .
<b>Exceções:</b>	

<b>Nome:</b>	carregaLancamentos()
<b>Responsabilidades:</b>	Carregar a lista de lançamentos da conta corrente.
<b>Pré-condições:</b>	Existe uma <i>contaCorrente</i> .
<b>Resultados:</b>	Retorna <i>data</i> , <i>extrato</i> , <i>tipoLancamento</i> , <i>descricao</i> , <i>numeroCheque</i> , <i>entrada</i> , <i>saida</i> de todos os <i>Lancamento</i> da <i>contaCorrente</i> .

<b>Nome:</b>	enviaLancamento(data: Date, extrato: boolean, tipoLancamento: Enumeration, descrição: String, numeroCheque: int, entrada: Currency, saída: Currency)
<b>Responsabilidades:</b>	Inserir um novo lançamento para a conta corrente com os parâmetros passados.
<b>Pré-condições:</b>	Existe uma <i>contaCorrente</i> .
<b>Pós-condições:</b>	Foi criado um <i>Lancamento</i> . Os atributos <i>data</i> , <i>descricao</i> , <i>entrada</i> , <i>saida</i> , <i>extrato</i> e <i>tipoLancamento</i> do <i>Lancamento</i> foram alterados para os valores dos parâmetros <i>data</i> , <i>descricao</i> , <i>entrada</i> , <i>saida</i> , <i>extrato</i> e <i>tipoLancamento</i> . Foi criada uma associação do novo <i>Lancamento</i> com a <i>contaCorrente</i> .
<b>Exceções:</b>	

## A.5 Diagramas de Colaboração

Aqui são mostrados os diagramas de colaboração referentes aos contratos mostrados na seção A.4.

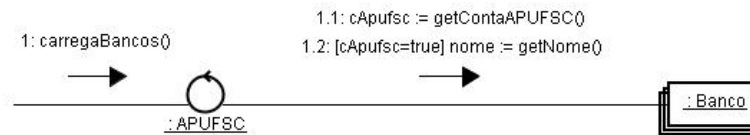


Figura 11: Diagrama de colaboração referente ao contrato `carregaBancos`.

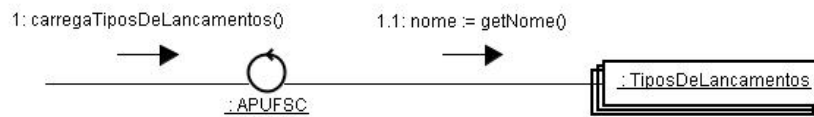


Figura 12: Diagrama de colaboração referente ao contrato `carregaTiposDeLancamentos`.

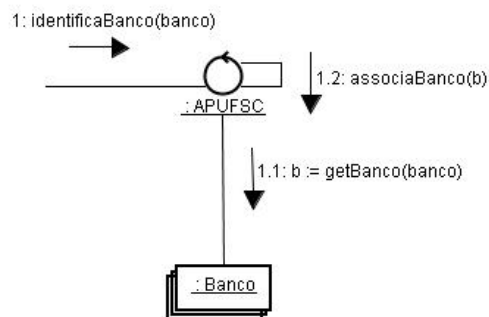


Figura 13: Diagrama de colaboração referente ao contrato `identificaBanco`.

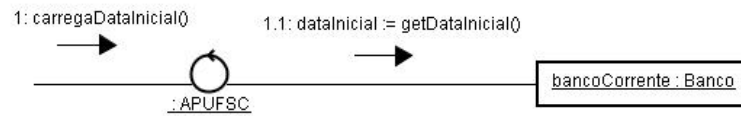


Figura 14: Diagrama de colaboração referente ao contrato `carregaDataInicial`.

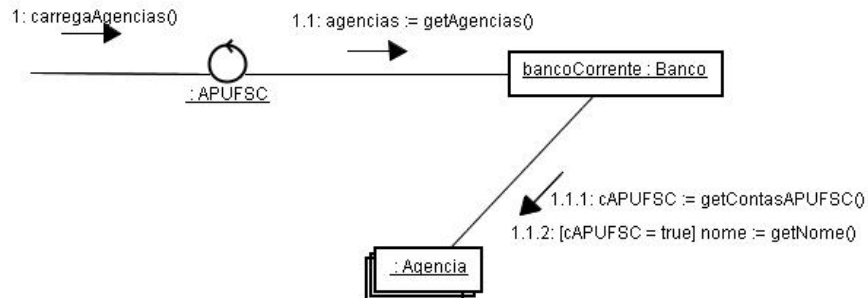


Figura 15: Diagrama de colaboração referente ao contrato `carregaAgencias`.

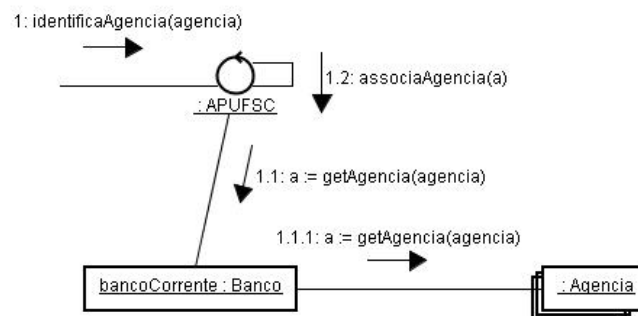


Figura 16: Diagrama de colaboração referente ao contrato `identificaAgencia`.

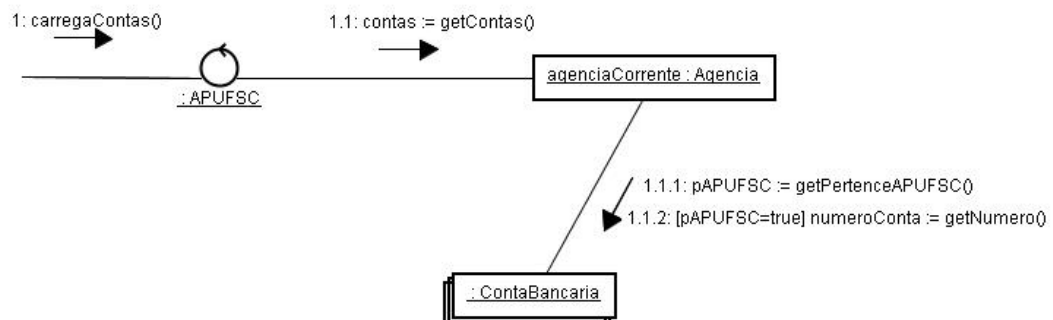


Figura 17: Diagrama de colaboração referente ao contrato `carregaContas`.



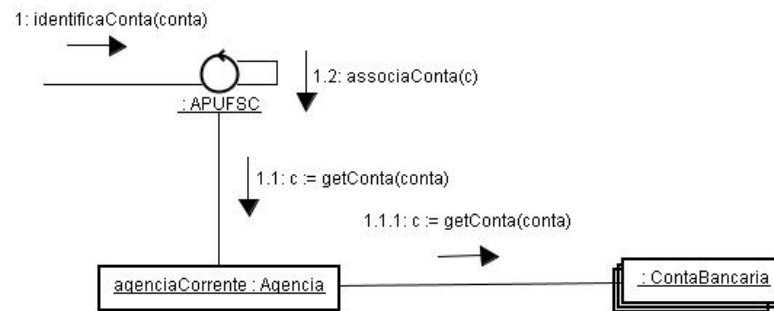


Figura 18: Diagrama de colaboração referente ao contrato `identificaConta`.

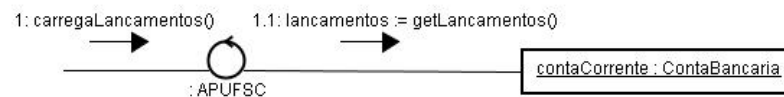


Figura 19: Diagrama de colaboração referente ao contrato `carregaLancamentos`.

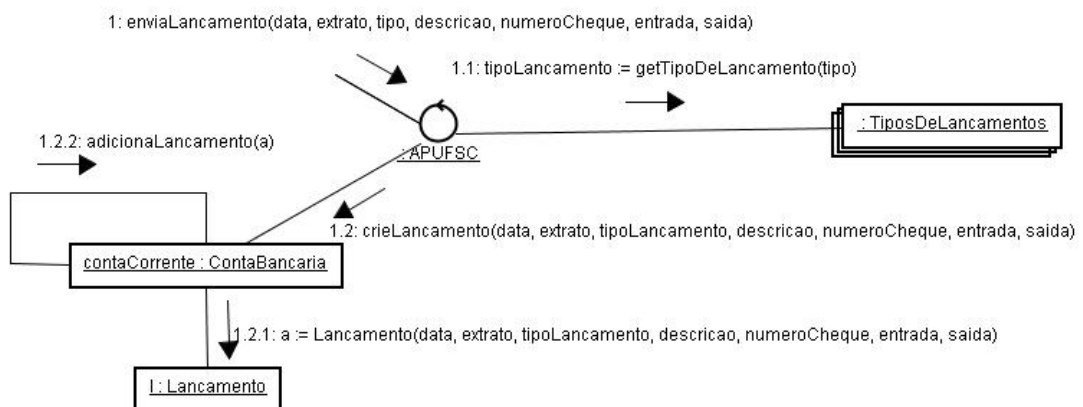


Figura 20: Diagrama de colaboração referente ao contrato `enviaLancamento`.

## A.6 Diagrama de Classes de Projeto

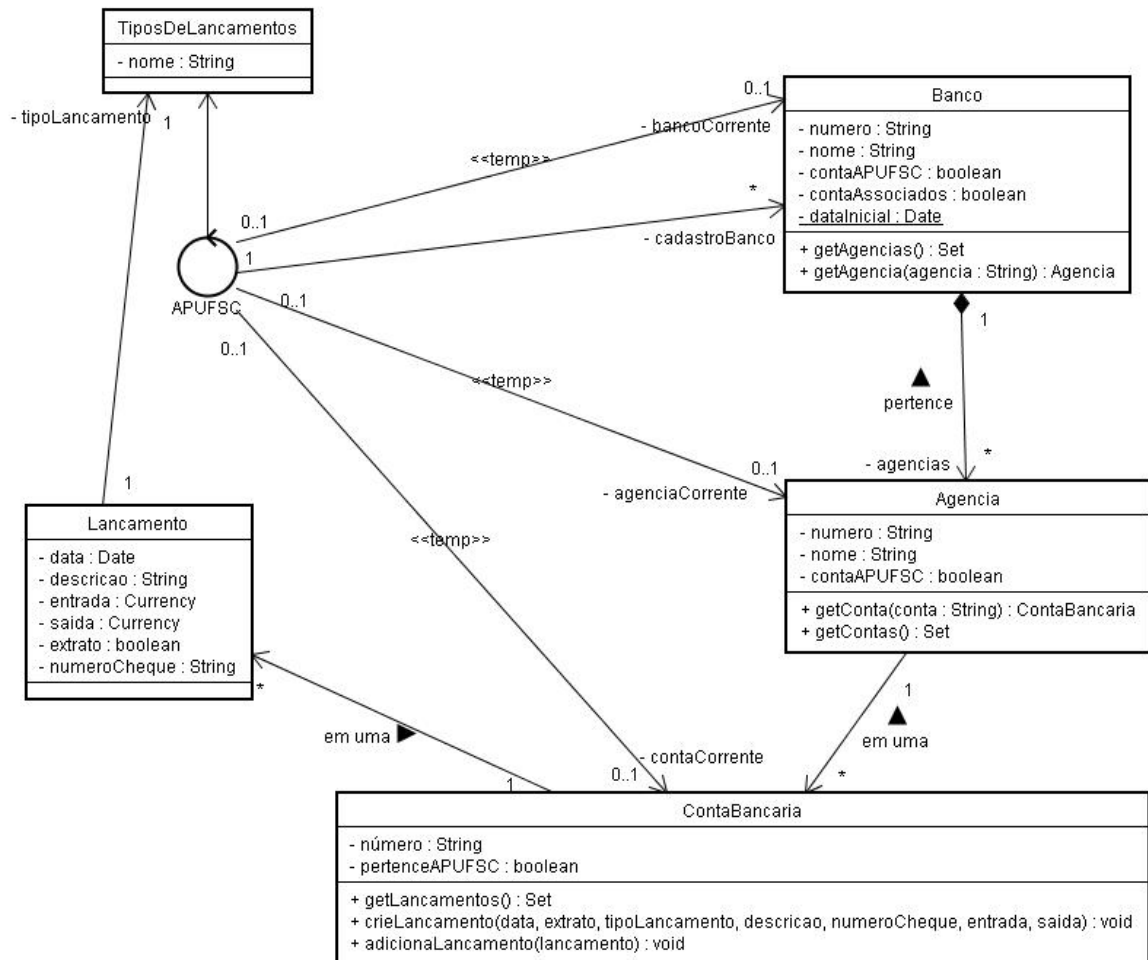


Figura 21: Diagrama de classes de projeto com as classes referentes ao caso de uso Lançar Receitas e Despesas.

## A.7 Implementação da Camada de Domínio

Aqui são exibidas as classes da camada de domínio resultantes do processo de reengenharia deste estudo de caso. As classes aqui apresentadas refletem apenas o caso de uso estudo, ou seja, alguns atributos e métodos referentes a outros casos de uso não foram implementados.

### A.7.1 Classe TiposDeLancamentos

```

public class TiposDeLancamentos {
    private String nome;

```

```

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}
}

```

## A.7.2 Classe Lancamento

```

import java.util.*;

public class Lancamento {
    private Date data;
    private String descricao;
    private Currency entrada;
    private Currency saida;
    private boolean extrato;
    private String numeroCheque;
    private TiposDeLancamentos tipoLancamento;

    public Lancamento (Date data, boolean extrato, TiposDeLancamentos tipoLancamento,
        String descricao, String numeroCheque, Currency entrada,
        Currency saida) {
        this.data = data;
        this.descricao = descricao;
        this.entrada = entrada;
        this.saida = saida;
        this.extrato = extrato;
        this.numeroCheque = numeroCheque;
        this.tipoLancamento = tipoLancamento;
    }

    public Date getData() {
        return data;
    }

    public String getDescricao() {
        return descricao;
    }

    public Currency getEntrada() {
        return entrada;
    }

    public Currency getSaida() {
        return saida;
    }

    public boolean getExtrato() {
        return extrato;
    }
}

```

```

    }

    public TiposDeLancamentos getTipoLancamento() {
        return tipoLancamento;
    }

    public String getNumeroCheque() {
        return numeroCheque;
    }

    public void setData(Date data) {
        this.data = data;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public void setEntrada(Currency entrada) {
        this.entrada = entrada;
    }

    public void setSaida(Currency setSaida) {
        this.saida = saida;
    }

    public void setExtrato(boolean extrato) {
        this.extrato = extrato;
    }

    public void setTipoLancamento(TiposDeLancamentos tipoLancamento) {
        this.tipoLancamento = tipoLancamento;
    }

    public void setNumeroCheque(String numeroCheque) {
        this.numeroCheque = numeroCheque;
    }
}

```

### A.7.3 Classe ContaBancaria

```

import java.util.*;

public class ContaBancaria {
    private String numero;
    private boolean pertenceAPUFSC;
    private Vector lancamentos;

    public String getNumero() {
        return numero;
    }

    public boolean getPertenceAPUFSC() {

```

```

        return pertenceAPUFSC;
    }

    public Vector getLancamentos() {
        return lancamentos;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }

    public void setPertenceAPUFSC(boolean pertenceAPUFSC) {
        this.pertenceAPUFSC = pertenceAPUFSC;
    }

    public void adicionaLancamento(Lancamento lancamento) {
        lancamentos.add(lancamento);
    }

    public void removeLancamento(Lancamento lancamento) {
        lancamentos.remove(lancamento);
    }

    public void criaLancamento(Date data, boolean extrato, TiposDeLancamentos tipoLancamento,
                                String descricao, String numeroCheque, Currency entrada,
                                Currency saida) {
        Lancamento lancamento = new Lancamento (data, extrato, tipoLancamento, descricao,
                                                    numeroCheque, entrada, saida);

        this.adicionaLancamento(lancamento);
    }
}

```

## A.7.4 Classe Agencia

```

import java.util.*;

public class Agencia {
    private String numero;
    private String nome;
    private boolean contaAPUFSC;
    private Vector contasBancarias;

    public String getNumero() {
        return numero;
    }

    public String getNome() {
        return nome;
    }

    public boolean getContasAPUFSC() {
        return contaAPUFSC;
    }
}

```

```

public Vector getContas() {
    Vector contas = new Vector();
    ContaBancaria contaAtual;
    boolean pAPUFSC;
    contasBancarias.trimToSize();

    for (int i = 0; i < contasBancarias.size(); i++) {
        contaAtual = (ContaBancaria) contasBancarias.elementAt(i);
        pAPUFSC = contaAtual.getPertenceAPUFSC();
        if (pAPUFSC) {
            contas.add(contaAtual.getNumero());
        }
    }
    return contas;
}

public ContaBancaria getConta(String conta) {
    contasBancarias.trimToSize();
    String contaAtual;

    for (int i = 0; i < contasBancarias.size(); i++) {
        contaAtual = ((ContaBancaria) contasBancarias.elementAt(i)).getNumero();
        if (conta.equals(contaAtual)) {
            return ((ContaBancaria) contasBancarias.elementAt(i));
        }
    }
    return null;
}

public void setNumero(String numero) {
    this.numero = numero;
}

public void setNome(String nome) {
    this.nome = nome;
}

public void setContaAPUFSC(boolean contaAPUFSC) {
    this.contaAPUFSC = contaAPUFSC;
}

public void adicionaConta(ContaBancaria conta) {
    contasBancarias.add(conta);
}

public void removeConta(ContaBancaria conta) {
    contasBancarias.remove(conta);
}
}

```

## A.7.5 Classe Banco

```

import java.util.*;

public class Banco {
    private String numero;
    private String nome;
    private boolean contaAPUFSC;
    private boolean contaAssociados;
    private static Date dataInicial;
    private Vector agencias;

    public String getNumero() {
        return numero;
    }

    public String getNome() {
        return nome;
    }

    public boolean getContaAPUFSC() {
        return contaAPUFSC;
    }

    public boolean getContaAssociados() {
        return contaAssociados;
    }

    public Date getDataInicial() {
        return dataInicial;
    }

    public Vector getAgencias() {
        Vector nomes = new Vector();
        Agencia agenciaAtual;
        boolean cAPUFSC;
        agencias.trimToSize();

        for (int i = 0; i < agencias.size(); i++) {
            agenciaAtual = (Agencia) agencias.elementAt(i);
            cAPUFSC = agenciaAtual.getContasAPUFSC();
            if (cAPUFSC) {
                nomes.add(agenciaAtual.getNome());
            }
        }
        return nomes;
    }

    public Agencia getAgencia(String agencia) {
        agencias.trimToSize();
        String agenciaAtual;

        for (int i = 0; i < agencias.size(); i++) {
            agenciaAtual = ((Agencia) agencias.elementAt(i)).getNumero();

```

```

        if (agencia.equals(agenciaAtual)) {
            return ((Agencia) agencias.elementAt(i));
        }
    }
    return null;
}

public void setNumero(String numero) {
    this.numero = numero;
}

public void setNome(String nome) {
    this.nome = nome;
}

public void setContaAPUFSC(boolean contaAPUFSC) {
    this.contaAPUFSC = contaAPUFSC;
}

public void setContaAssociados(boolean contaAssociados) {
    this.contaAssociados = contaAssociados;
}

public void setDataInicial(Date dataInicial) {
    this.dataInicial = dataInicial;
}

public void adicionaAgencia(Agencia agencia) {
    agencias.add(agencia);
}

public void removeAgencia(Agencia agencia) {
    agencias.remove(agencia);
}
}

```

## A.7.6 Classe APUFSC

```

import java.util.*;

public class APUFSC {
    private Vector cadastroBancos;
    private Vector tiposDeLancamentos;

    private ContaBancaria contaCorrente;
    private Banco bancoCorrente;
    private Agencia agenciaCorrente;

    public Vector carregaBancos() {
        Vector nomes = new Vector();
        Banco bancoAtual;
        boolean cAPUFSC;
        cadastroBancos.trimToSize();
    }
}

```



```

    for (int i = 0; i < cadastroBancos.size(); i++) {
        bancoAtual = (Banco) cadastroBancos.elementAt(i);
        cAPUFSC = bancoAtual.getContaAPUFSC();
        if (cAPUFSC) {
            nomes.add(bancoAtual.getNome());
        }
    }
    return nomes;
}

public Vector carregaTiposDeLancamentos() {
    Vector nomes = new Vector();
    TiposDeLancamentos tipoAtual;
    tiposDeLancamentos.trimToSize();

    for (int i = 0; i < tiposDeLancamentos.size(); i++) {
        tipoAtual = (TiposDeLancamentos) tiposDeLancamentos.elementAt(i);
        nomes.add(tipoAtual.getNome());
    }

    return nomes;
}

public void identificaBanco(String banco) {
    Banco b = getBanco(banco);
    associaBanco(b);
}

public Banco getBanco (String nome) {
    cadastroBancos.trimToSize();
    String bancoAtual;

    for (int i = 0; i < cadastroBancos.size(); i++) {
        bancoAtual = ((Banco) cadastroBancos.elementAt(i)).getNumero();
        if (cadastroBancos.equals(bancoAtual)) {
            return ((Banco) cadastroBancos.elementAt(i));
        }
    }
    return null;
}

public void associaBanco(Banco b) {
    bancoCorrente = b;
}

public Date carregaDataInicial() {
    return bancoCorrente.getDataInicial();
}

public Vector carregaAgencias() {
    return bancoCorrente.getAgencias();
}

```



# *Referências*

- [ALC 02] ALCÂNTARA, R. L. **Sobre o Desenvolvimento de Sistemas de Informação Utilizando Software Livre - Uma Experiência no Serviço Público**. Florianópolis: Universidade Federal de Santa Catarina, Novembro, 2002. Dissertação de Mestrado.
- [ANT ] ANTUNES, L.; ALMEIDA, A.; PEREIRA, N. **Reengenharia**. Disponível em: <http://eden.dei.uc.pt/gestao/forum/temas/classicos/reengenharia.html>. Acessado em 15/09/04.
- [BAR 02] BARTIÉ, A. **Garantia na Qualidade de Software**. Rio de Janeiro: Editora Campus, 2002.
- [BER 99] BERGEY, J. et al. Why reengineering projects fail. Pittsburgh, PA: Software Engineering Institute - Carnegie Mellon University, Abril, 1999. Relatório TécnicoCMU/SEI-99-TR-010.
- [BOO 98] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Addison Wesley, 1998.
- [cas 88] Case tools for reverse engineering. **CASE Outlook, CASE Consulting Group**, [S.l.], v.2, n.2, p.1–15, 1988.
- [dAC 03] DE ABREU CYBIS, W. **Engenharia de Usabilidade: Uma Abordagem Ergonômica**. Disponível em: [http://www.labiutil.inf.ufsc.br/Apostila\\_nvVersao.pdf](http://www.labiutil.inf.ufsc.br/Apostila_nvVersao.pdf). Acessado em 03/07/04.
- [DAV 93] DAVENPORT, T. H. **Process Innovation, Reengineering Work through Information Technology**. Boston: Harvard Business School Press, 1993.
- [DEI 01] DEITEL, H. M.; DEITEL, P. J. **Java, como programar**. Porto Alegre: Bookman, 2001.
- [dPPF 03] DE PÁDUA PAULA FILHO, W. **Engenharia de Software: Fundamentos, Métodos e Padrões**. 2. ed. Rio de Janeiro: LTC, 2003.
- [FLY 93] FLYNN, K. Critical success factors for a successful business reengineering project. **CASE World Conference Proceedings**, Boston, Outubro, 1993.
- [HAM 90] HAMMER, M. Reengineer work: Don't automate, obliterate. **Harvard Business Review**, [S.l.], p.104–112, July-August, 1990.
- [HAM 93] HAMMER, M.; CHAMPY, J. **Reengineering the Corporation: A Manifesto for Business Revolution**. New York: North River Press Inc., 1993.
- [JAC 94] JACOBSON, I.; ERICSSON, M.; JACOBSON, A. **The Object Advantage: Business Process Reengineering with Object Technology**. New York: ACM Press, 1994.
- [KRU 03] KRUCHTEN, P. **Using the RUP to Evolve a Legacy System**. Disponível em: <http://www-106.ibm.com/developerworks/rational/library/389.html>. Acessado em 03/07/04.
- [LAR 00] LARMAN, C. **Utilizando UML e Padrões**. Porto Alegre: Bookman, 2000.
- [LEM ] LEMOS, C. M. S. **Reengenharia de Sistemas de Informação: Uma abordagem de implementação**. Trabalho desenvolvido ao longo da Pós Graduação de Sistemas de Informação do ISEG.
- [MAR 02] MARTINS, J. C. C. **Gestão de Projetos de Desenvolvimento de Software (PMI-UML)**. Rio de Janeiro: Brasport, 2002.

- [MER 93] MERLO, E.; ET AL. Reverse engineering of user interfaces. **Working Conference on Reverse Engineering**, IEEE, Baltimore, p.171–178, Maio, 1993.
- [MIC 03] MICROSOFT. **Visão geral do Access 2003**. Disponível em: <http://www.microsoft.com/brasil/office/access/overview.asp>. Acessado em 13/10/04.
- [OSB 90] OSBORNE, W. M.; CHIKOFFSKY, E. J. Fitting pieces to the maintenance puzzle. **IEEE Journal**, [S.l.], p.10–11, Janeiro, 1990.
- [PRE 00] PRESSMAN, R. S. **Software Engineering: A Pratictioner´s Approach**. 5. ed. McGraw-Hill Education, 2000.
- [RAT 98] RATIONAL. **Rational Unified Process: Best Pratices for Software Development Teams**. Disponível em: <http://www-106.ibm.com/developerworks/rational/library/253.html>. Acessado em 03/07/04.
- [SCH 04] SCHÜRHAUS, S.; REMÁCULO, L. P. **Rational Unified Process**. Trabalho realizado para a disciplina de Engenharia de Software. Ciências da Computação, UFSC, Florianópolis, 2004. Disponível em: <http://www.inf.ufsc.br/~sabras>. Acessado em 23/06/04.
- [WAZ 04] WAZLAWICK, R. S. **Análise e Projeto de Sistemas de Informação Orientados a Objetos**. Rio de Janeiro: Elsevier Editora, 2004.