MARCOS VINÍCIUS SOARES

DESENVOLVIMENTO E MANUTENÇÃO DE PRODUTOS DE SOFTWARE: OS CASOS CPPD E CIS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS – BRASIL
2006

MARCOS VINÍCIUS SOARES

DESENVOLVIMENTO E MANUTENÇÃO DE PRODUTOS DE SOFTWARE: OS CASOS CPPD E CIS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:
Engenharia e Qualidade de Software

Orientador:
Prof. Dr. Heitor Augustus Xavier Costa

LAVRAS
MINAS GERAIS – BRASIL
2006

Ficha Catalográfica preparada pela Divisão de Processos Técnico da Biblioteca Central da UFLA

Soares, Marcos Vinícius

Desenvolvimento e Manutenção de Produtos de Software: Os Casos CPPD e CIS / Marcos Vinícius Soares. Lavras – Minas Gerais, 2006, 140p.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação

1. Desenvolvimento de Produtos de Software. 2. Manutenção de Produtos de Software. 3. UML. I. SOARES, M. V. II. Universidade Federal de Lavras. III. Título.

MARCOS VINÍCIUS SOARES

DESENVOLVIMENTO E MANUTENÇÃO DE PRODUTOS DE SOFTWARE: OS CASOS CPPD E CIS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 25 de Abril de 2006

_____

Prof. Dr. André Luiz Zambalde

_____

Profª. MSc. Olinda Nogueira Paes Cardoso

_____

Prof. Dr. Heitor Augustus Xavier Costa
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

Ao meu pai, Antônio Alves Soares; minha mãe, Ana Maria Leite Soares; meu irmão Rubens; minha irmã Natália e aos demais familiares pelo apoio, incentivo e confiança.

AGRADECIMENTOS

A Deus, pela vida e força para a realização deste trabalho.

À minha família, pelo carinho e apoio incondicional em todos os momentos, minha eterna gratidão.

Ao meu pai, Antônio Alves Soares, pelo exemplo de profissionalismo e por todas as palavras de incentivo que sempre me fizeram prosseguir.

À minha mãe, Ana Maria Leite Soares, por todo o apoio, carinho e força durante toda minha vida.

Ao Professor Heitor Augustus Xavier Costa, meu orientador, por essa oportunidade, amizade, apoio e conselhos durante a realização deste trabalho.

A todos os professores do Departamento de Ciência da Computação, cujos ensinamentos tornaram possível o meu crescimento profissional.

Aos demais colegas do Departamento de Ciência da Computação que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Muito obrigado!

RESUMO

Desenvolvimento e Manutenção de Produtos de Software: Os Casos CPPD e CIS

A Diretoria de Recursos Humanos da Universidade Federal de Lavras possui a função de manter um cadastro dos servidores da universidade, controlar o regime de trabalho e avaliar o desempenho de cada servidor para progressão funcional. Atualmente, existe um cadastro de servidores armazenados de forma precária, com dados em planilhas e documentos impressos. O objetivo deste trabalho é organizar os dados relativos a pessoal através da manutenção do produto de software CPPD (Comissão Permanente de Pessoal Docente) e o desenvolvimento de um novo produto, denominado CIS (Comissão Interna de Supervisão). A manutenção será realizada com base nas técnicas de Engenharia Reversa e Reestruturação. O desenvolvimento será baseado na metodologia proposta pela UML e pelo Modelo de Desenvolvimento Iterativo e Incremental. Com a implantação dos novos produtos de software ocorrerão melhorias nos relatórios além de facilitar o cadastro e o controle do pessoal.

Palavras-chave: Desenvolvimento de Produtos de Software, Manutenção de Produtos de Software, UML.

ABSTRACT

Software Development and Maintenance: the cases CPPD and CIS

The Management of Human Resources of the Federal University of Lavras has the following functions: to maintain a directory of the staff members of the university, to control the working regime and to evaluate each member of the staff for career progression. Actually the directory is set up, but the data are in rough notes and printed documents, so in bad storage. The objective of this work is to organize the staff members data through the maintenance of the software CPPD (Comissão Permanente de Pessoal Docente) and the development of a new software, called CIS (Comissão Interna de Supervisão). The maintenance will be accomplished based in the techniques of Reverse Engineering and Restructuring. The development will be based on the methodology proposed by UML and the Incremental Model. With these new softwares the filling in of the information in the staff directory, staff control, reports and draw out informations will be improved.

Keywords: Software Development, Software Maintenance, UML.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

A dependência e a demanda crescentes da sociedade em relação à Tecnologia da Informação e, em particular, a produtos de software, têm ressaltado uma série de problemas relacionados ao processo de desenvolvimento destes produtos: alto custo, alta complexidade, dificuldade de manutenção, e uma disparidade entre as necessidades dos usuários e o produto desenvolvido (DEMARCO, 1995).

Estes problemas afligem a área de software desde sua criação. PRESSMAN (2002) os identifica como uma aflição crônica, não uma crise pontual. Para administrá-los, a comunidade de Engenharia de Software, ao longo dos últimos 30 anos, estuda e implementa práticas de desenvolvimento e manutenção de produtos de software bem organizados e documentados.

O presente trabalho descreve a manutenção do produto de software CPPD e o desenvolvimento do produto de software CIS. Entende-se por desenvolvimento um conjunto de atividades cuja meta é a produção de software (SOMMERVILE, 2003). A manutenção é a atividade durante a qual ocorrem modificações no produto de software, buscando mantê-lo disponível, corrigir suas falhas, melhorar seu desempenho e adequá-lo aos requisitos novos ou modificados, conforme as necessidades de seus usuários (ANSI/IEEE, 1983).

1.1 Motivação

Constatou-se a real necessidade, junto a Diretoria de Recursos Humanos da Universidade Federal de Lavras, da manutenção do produto de software CPPD e o desenvolvimento de um novo produto de software denominado CIS. Ambos visando facilitar o cadastro e o controle dos demais processos funcionais dos docentes e dos técnicos administrativos da universidade, possibilitando obter informações relevantes através de relatórios mais elaborados. Os dois produtos de software são classificados como Sistemas de Informação.

Segundo STAIR (2002), os Sistemas de Informação, independente do seu nível ou classificação, têm como maior objetivo: auxiliar os processos de tomada de decisões na empresa. Um Sistema de Informação eficiente pode ter um grande impacto na estratégia corporativa e no sucesso empresarial. Este impacto pode beneficiar a empresa, os clientes e/ou usuários e qualquer indivíduo ou grupo que interagir com os Sistemas de Informação.

1.2 Objetivos

Os objetivos do projeto são a manutenção do produto de software CPPD e o desenvolvimento do produto de software CIS, que têm por principal finalidade automatizar os processos realizados sob as informações cadastrais dos docentes e dos técnicos administrativos da Universidade Federal de Lavras, bem como controlar os regimes de trabalho e avaliar seus desempenhos para progressão funcional. Os produtos de software devem atender às solicitações advinhas da Diretoria de Recursos Humanos da Universidade Federal de Lavras.

São objetivos específicos deste projeto:

- Facilitar a extração de informações estatísticas e gerenciais, contribuindo com a geração de indicadores e melhorando o acesso aos dados dos servidores da universidade;
- Melhorar a qualidade da informação gerada, contribuindo com a eliminação das redundâncias e oferecendo transparência e confiabilidade aos dados;
- Aumentar a disponibilidade e a agilidade atendendo as demandas de informação;
- Viabilizar a execução dos sistemas remotamente, para que, futuramente, outros órgãos e departamentos da universidade tenham acesso aos dados.

1.3 Tecnologia Utilizada

A modelagem dos produtos de software foi realizada utilizando a ferramenta JUDE/*Community* 1.6.2, por ser gratuita, simples e atender as necessidades do trabalho. Além disso, o seu suporte é gratuito e as dúvidas são esclarecidas rapidamente pela equipe de desenvolvimento.

Foi necessário configurar um servidor de banco de dados com o objetivo de armazenar os dados pessoais, o regime de trabalho e as progressões dos docentes e técnicos administrativos. O SGBD (Sistema Gerenciador de Banco de Dados) escolhido foi o MySQL 4.1.12, pois possui alta confiabilidade, alta performance e flexibilidade, além de ser o banco de dados *open-source* mais conhecido mundialmente (MYSQL, 2005). A administração dos bancos de dados é simples e fácil, sendo utilizado o MySQL *Control Center* 0.9.4-beta.

Para o desenvolvimento dos produtos de software, foi utilizada a tecnologia J2SE (*Java Standard Edition*) 1.4.2_08. O uso desta tecnologia se justifica pela portabilidade,

robustez e, segundo JAVA (2005), por fornecer a base para uma conexão segura a diversos SGBD's, incluindo o MySQL.

O ambiente de desenvolvimento *Sun ONE Studio 5 Standart Edition* foi utilizado para edição do código-fonte e construção da interface gráfica do sistema. Segundo SUN (2005), *Sun ONE Studio* fornece um compreensivo e altamente otimizado ambiente de desenvolvimento para criação dos mais diversos tipos de produtos de software.

Para criação e compilação dos relatórios foi utilizado o iReport 0.5.1. Segundo IREPORT (2005), iReport é uma ferramenta robusta, intuitiva e possui alto grau de usabilidade de interface. Além disso, é compatível com JasperReports, que foi utilizado para emissão dos relatórios (versão 1.01). JasperReports é uma ferramenta gratuita, escrita em Java e possui a habilidade de exibir um rico conteúdo na tela ou em diversos formatos de arquivos, sendo possível imprimi-los posteriormente (JASPER, 2005).

1.4 Estrutura do Trabalho

Este trabalho está organizado em seis capítulos. O Capítulo 2 expõe a importância do desenvolvimento de produtos de software e, em seguida, apresenta modelos de processo de desenvolvimento. O Capítulo 3 descreve, com base na literatura existente, a importância, as dificuldades, as tecnologias utilizadas e os modelos de processo de manutenção de produtos de software. O Capítulo 4 apresenta a história da UML, descrevendo, posteriormente, sua importância e os diagramas a ela relacionados. O Capítulo 5 apresenta a modelagem de dados utilizado o Modelo Entidade Relacionamento. O Capítulo 6 descreve o projeto proposto, abordando o processo de desenvolvimento empregado nos produtos de software CPPD e CIS. O Capítulo 7 apresenta a conclusão, a contribuição e os trabalhos futuros.

2 DESENVOLVIMENTO DE PRODUTOS DE SOFTWARE

2.1 Considerações Iniciais

Em 1968, uma conferência foi organizada para discutir a chamada crise do software que resultava diretamente da introdução (naquela época) de hardware de computador de terceira geração. A capacidade deste hardware tornava viáveis aplicações de computador até então inimagináveis. O produto de software resultante era bem maior e mais complexo que os produtos de software precedentes (SOMMERVILLE, 2003).

A experiência inicial de construção desses produtos de software mostrou que uma abordagem informal para o seu desenvolvimento não era o bastante. Projetos importantes sofriam atrasos, às vezes, de alguns anos. Por apresentarem custos maiores do que os inicialmente previstos, produtos de software não eram confiáveis, eram de difícil manutenção e tinham desempenho inferior ao esperado. O desenvolvimento de produtos software estava em crise. Os custos de hardware caiam, enquanto os de produtos de software subiam rapidamente. Novas técnicas e novos métodos eram necessários para controlar a complexidade inerente aos produtos de software (SOMMERVILLE, 2003).

Houve um grande progresso desde 1968 e, atualmente, tem-se uma compreensão muito melhor das atividades envolvidas no desenvolvimento de produtos de software. São desenvolvidos processos e métodos eficazes de especificação, projeto e implementação de produtos de software. Novas notações e ferramentas reduzem o esforço necessário para produzir produtos de software grandes e complexos.

Este capítulo destaca a importância do processo de software, definido por SOMMERVILLE (2003) como um conjunto de atividades, cuja meta é o desenvolvimento e/ou a evolução do produto de software. Em seguida, são apresentados alguns modelos de processo de desenvolvimento de produtos de software.

2.2 Importância de Processo de Desenvolvimento

O produto de software tornou-se força motora que dirige a tomada de decisões nos negócios. Ele serve de base à moderna investigação científica e às soluções de problemas de engenharia, bem como é um fator chave que diferencia os produtos e serviços modernos. Ele está embutido em sistemas de todas as naturezas: de transportes, médicos, de telecomunicações, militares, de processos industriais, de escritório entre outros. O

produto de software é virtualmente inevitável no mundo moderno, pois, à medida que se entra no vigésimo primeiro século, ele se tornar um motor para novos avanços em tudo, da educação elementar à engenharia genética (PRESSMAN, 2002).

Direta ou indiretamente, o produto de software afeta todas as pessoas do mundo industrializado. Assim sendo, torna-se essencial a aplicação do processo adequado para desenvolvimento de produtos de software assegurando qualidade, confiabilidade e um bom custo-benefício.

2.3 Modelos de Processo de Desenvolvimento

Um modelo de processo de desenvolvimento é uma descrição simplificada de um processo de desenvolvimento de produtos de software, que é apresentada a partir de uma perspectiva existente (SOMMERVILE, 2003). Os modelos, pela sua natureza, são simplificações; assim, um modelo de processo de desenvolvimento de produtos de software é uma abstração do processo real que está sendo descrito.

Nas subseções a seguir, são apresentados alguns modelos de processo de desenvolvimento de produtos de software.

2.3.1 Modelo Cascata

Proposto por ROYCE (1970), o Modelo Cascata (Figura 2.1) prevê ciclos de *feedback,* sendo representado pelos seguintes estágios:

- **Análise e definição de requisitos**. As funções, as restrições e os objetivos do produto de software são estabelecidos por meio da consulta aos seus usuários. Em seguida, são definidos em detalhes e servem como sua especificação;
- **Projetos do sistema e do produto de software**. O processo de projeto de sistemas agrupa os requisitos em sistemas de hardware e de produto de software. Ele estabelece uma arquitetura geral do sistema. O projeto do produto de software envolve a identificação e a descrição das abstrações fundamentais do produto de software e suas relações;
- **Implementação e teste de unidades**. Durante esse estágio, o projeto do produto de software é compreendido como um conjunto de programas ou unidades de programa. O teste de unidades verifica se cada unidade atende a sua especificação;

- **Integração e teste de sistemas**. As unidades de programa são integradas e testadas como um sistema completo a fim de garantir se os requisitos do produto de software foram atendidos. Depois dos testes, o produto de software é entregue ao cliente;
- **Operação e Manutenção**. Normalmente, embora não necessariamente, esta é a fase mais longa do ciclo de vida. O produto de software é instalado e colocado em operação. A manutenção consiste em corrigir erros que não foram descobertos em estágios anteriores do ciclo de vida, melhorando a implementação das unidades e/ou aumentando as funções à medida que novos requisitos são descobertos.

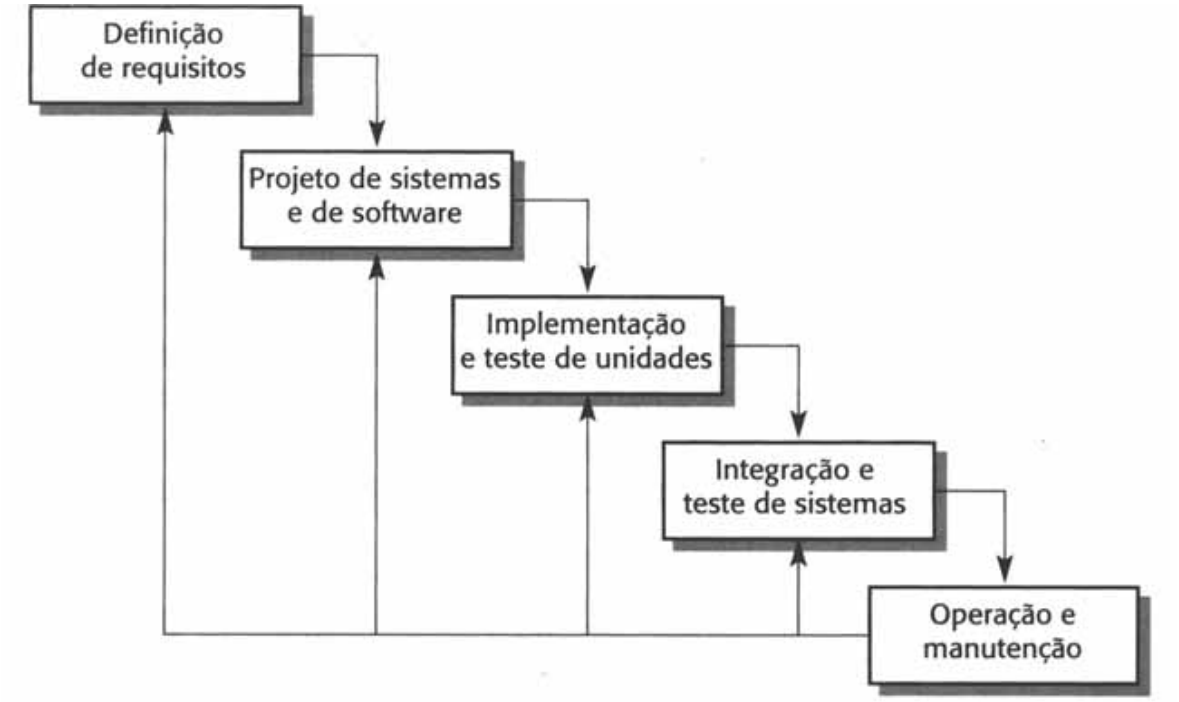


Figura 2.1 – Modelo Cascata (Fonte: SOMMERVILLE, 2003)

Cada uma das fases não deve iniciar até que a fase precedente tenha sido concluída. Na prática, esses estágios se sobrepõem e trocam informações entre si. Durante o projeto, são identificados problemas com os requisitos; durante a codificação são verificados problemas de projeto, e assim por diante. Este processo de desenvolvimento não é um modelo linear simples, pode envolver uma seqüência de iterações das atividades de desenvolvimento (SOMMERVILLE, 2003).

O Modelo Cascata é o mais antigo e é o paradigma para engenharia de software mais amplamente usado (PRESSMAN, 2002). No entanto, este paradigma levou seus atuais adeptos a questionar sua eficácia (HANNA, 1995). Entre os problemas que são algumas vezes encontrados, quando o Modelo Cascata é aplicado, estão:

- Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Apesar do modelo linear poder acomodar interação, o faz indiretamente. Como resultado, modificações podem causar confusão à medida que a equipe de projeto prossegue;
- Em geral, é difícil para o cliente estabelecer todos os requisitos explicitamente. O Modelo Cascata exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de vários projetos;
- O cliente precisa ter paciência. Uma versão do produto de software não vai ficar disponível até o projeto terminar. Um erro grosseiro pode ser desastroso, se não for detectado até que o produto de software seja revisto.

BRADAC *et al* (1994) descobriu que a natureza linear do Modelo Cascata leva a estados de bloqueio, nos quais alguns membros da equipe de projeto precisam esperar que outros membros completem as tarefas dependentes. Na realidade, o tempo de espera pode exceder o tempo de trabalho produtivo. O estado de bloqueio tende a ocorrer mais no início e no fim do processo.

Apesar de ter pontos fracos, o Modelo Cascata é significativamente melhor que uma abordagem aleatória para o desenvolvimento de produtos de software (PRESSMAN, 2002).

2.3.2 Modelo de Prototipagem

O Modelo de Prototipagem (Figura 2.2) começa com a definição dos requisitos. O engenheiro de software e o cliente encontram-se e definem os objetivos gerais do produto de software, identificam as necessidades conhecidas e delineiam áreas que necessitam de mais definições. Um projeto rápido é então realizado. Esse projeto parte de um protótipo, o qual é avaliado pelo cliente e utilizado para refinar os requisitos do produto de software que será desenvolvido. Interações ocorrem, à medida que o protótipo é ajustado, para satisfazer às necessidades do cliente, enquanto, ao mesmo tempo, permitem ao engenheiro de software entender melhor o que precisa ser feito (PRESSMAN, 2002).

Assim que o protótipo tiver cumprido sua finalidade, ele deve ser descartado, ou seja, não há alternativa senão começar tudo de novo e construir uma versão re-projetada, na qual os problemas identificados são resolvidos (BROOKS, 1975).

Segundo (PRESSMAN, 2002), tanto os clientes quanto os engenheiros de software gostam de utilizar o paradigma de prototipagem. Todavia, a prototipagem pode ser problemática pelas seguintes razões:

- O cliente vê o que parece ser uma versão executável do produto de software, ignorando que o protótipo apenas consegue funcionar precariamente, sem saber de que na pressa de fazê-lo rodar, ninguém considerou a sua qualidade global ou a questão de manutenibilidade em longo prazo. O cliente reclama e exige alguns consertos para transformar o protótipo em um produto de software final. Em geral, a gerência de desenvolvimento do produto de software concorda;
- O engenheiro de software freqüentemente faz concessões na implementação a fim de conseguir rapidamente um protótipo executável. Um sistema operacional ou uma linguagem de programação inapropriada pode ser usado simplesmente por estar disponível e serem conhecidos; um algoritmo ineficiente pode ser implementado simplesmente para demonstrar uma possibilidade. Passado um certo tempo, o engenheiro de software pode ficar familiarizado com essas escolhas e esquecer todas as razões por que elas eram inadequadas. A escolha muito aquém da ideal tornou-se agora parte integral do produto de software.

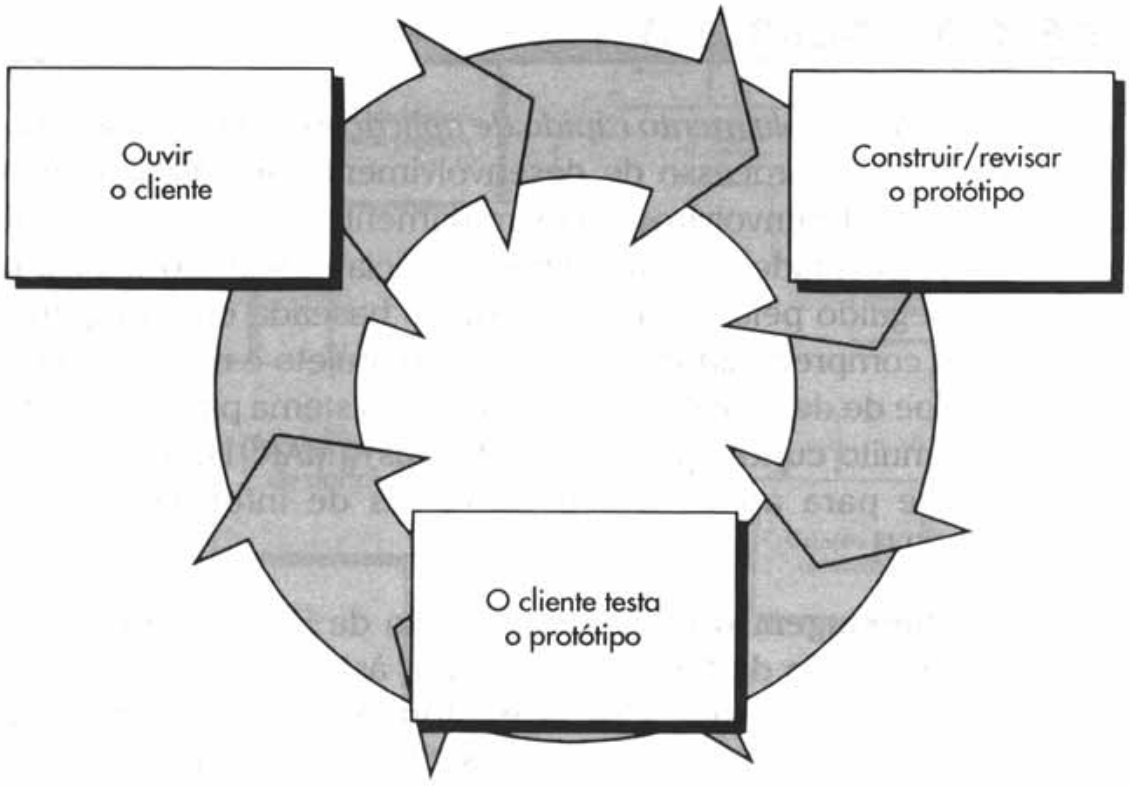


Figura 2.2 – Modelo de Prototipagem (Fonte: PRESSMAN, 2002)

A prototipagem pode ser um paradigma efetivo para o desenvolvimento de produtos de software, desde que o cliente e o engenheiro de software estejam de acordo que o protótipo a ser construído serve apenas como um mecanismo para definição dos requisitos, sendo descartado posteriormente para que o produto de software real seja submetido à engenharia com o objetivo de buscar qualidade e manutenibilidade.

2.3.3 Modelos de Processo de Software Evolucionários

Torna-se cada vez mais evidente que o produto de software, como todos os sistemas complexos, evolui durante um período de tempo (GILB, 1988). Requisitos mudam freqüentemente à medida que o desenvolvimento prossegue, dificultando um caminho direto para um produto final; prazos reduzidos de mercado tornam impossível completar um produto de software abrangente, mas uma versão reduzida pode ser elaborada para fazer face à competitividade ou às pressões do negócio. Nesse caso, e em situações semelhantes, os engenheiros de software precisam de um modelo de processo que tenha sido explicitamente projetado para acomodar um produto que evolui com o tempo (PRESSMAN, 2002).

A natureza evolutiva do produto de software não é considerada em nenhum dos modelos anteriormente apresentados.

Modelos evolucionários são interativos. Eles são caracterizados de forma a permitir aos engenheiros de software desenvolver versões cada vez mais completas do produto de software. As subseções a seguir apresentam alguns modelos evolucionários.

2.3.3.1 Modelo Incremental

A abordagem do modelo incremental, sugerida por MILLS *et al* (1980), combina elementos do Modelo Cascata com a filosofia interativa do Modelo de Prototipagem. Como mostra a Figura 2.3, um modelo incremental aplica seqüências lineares de uma forma racional à medida que o tempo passa. Cada seqüência produz um incremento factível do produto de software (MCDERMID & ROOK, 1993). Uma vez identificados os incrementos, os requisitos, para as funções a serem entregues no primeiro incremento, são identificados em detalhes e esse incremento é desenvolvido.

No primeiro incremento, freqüentemente chamado núcleo do produto de software, os requisitos básicos são satisfeitos, mas muitas características suplementares (algumas conhecidas, outras desconhecidas) deixam de ser elaboradas. Este núcleo é usado pelo

cliente (ou passa por revisão detalhada). Um plano é desenvolvido para o próximo incremento, como resultado do uso e/ou avaliação. O plano visa à modificação deste núcleo para melhor satisfazer às necessidades do cliente e a elaboração de características e funcionalidade adicional. Esse processo é repetido após a realização de cada incremento, até que o produto de software completo seja construído (PRESSMAN, 2002). SOMMERVILLE (2003) aponta algumas vantagens do modelo interativo:

- Os clientes não precisam esperar até que todo o produto de software seja entregue, para então tirarem proveito dele. O primeiro estágio satisfaz seus requisitos mais importantes e, assim, o produto de software pode ser imediatamente utilizado;
- Os clientes podem utilizar os primeiros incrementos como um protótipo e obter uma experiência que forneça os requisitos para estágios posteriores do produto de software;
- Existe um risco menor de fracasso do produto de software. Embora possam ser encontrados problemas em alguns incrementos, é provável que alguns incrementos sejam entregues com sucesso ao cliente;
- Como as funções prioritárias são entregues primeiro e incrementos posteriores são integrados a elas, é inevitável que as funções do produto de software mais importantes passem pela maior parte dos testes. Isso significa que é menos provável que os clientes encontrem falhas do produto de software na parte mais importante.

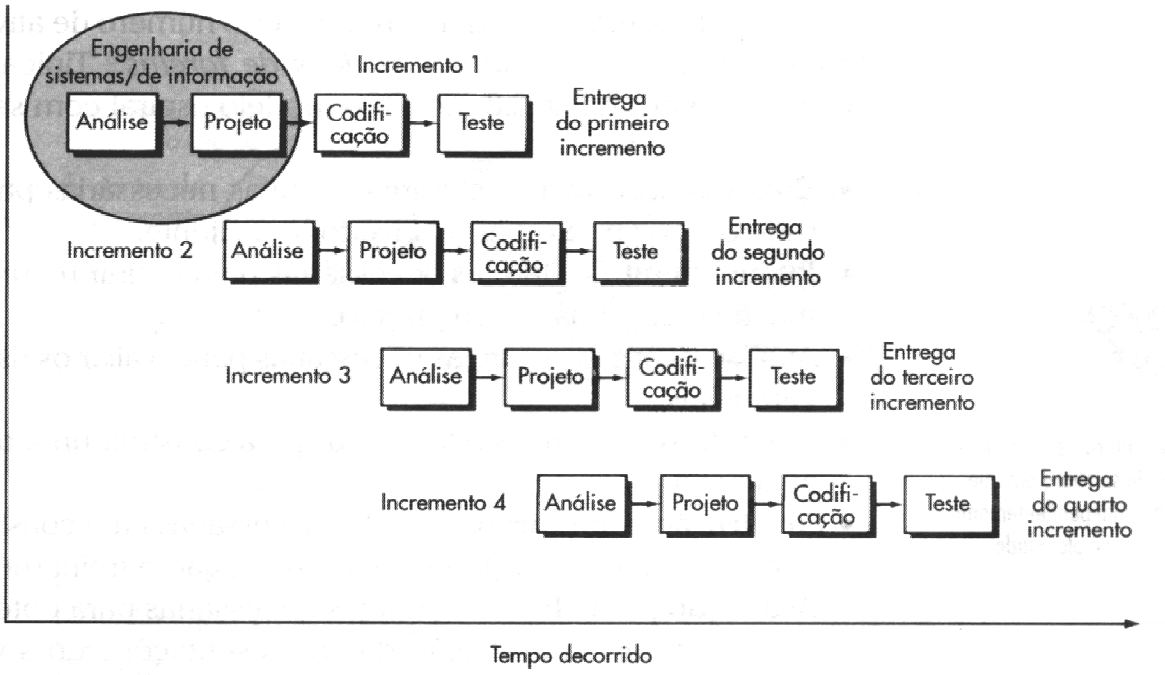


Figura 2.3 – Modelo Incremental (Fonte: PRESSMAN, 2002)

Contudo, existem alguns problemas com o desenvolvimento incremental. Os incrementos devem ser relativamente pequenos e cada incremento deve produzir alguma funcionalidade para o sistema. Sendo assim, pode ser difícil mapear os requisitos dos clientes nos incrementos de tamanho correto. Além disso, a maioria dos produtos de software exige um conjunto de facilidades básicas que são utilizadas por diferentes partes do sistema. Como os requisitos não são definidos em detalhes até que um incremento seja implementado, é difícil identificar facilidades comuns que todos os incrementos exijam (SOMMERVILLE, 2003).

Uma recente evolução dessa abordagem incremental, chamada de programação extrema (*extreme programming*), foi apresentada por BECK (1999). Ela tem como base o desenvolvimento e a entrega de incrementos pequenos, o envolvimento do cliente no processo, a constante melhoria de código e a programação impessoal. O artigo de BECK (1999) inclui vários relatórios de sucesso dessa abordagem, mas ainda é muito cedo para dizer se esta se tornará uma abordagem de uso dominante para o desenvolvimento de produtos de software.

2.3.3.2 Modelo Espiral

O Modelo Espiral, originalmente proposto por BOEHM (1988), é bastante conhecido. Em vez de representar o processo de desenvolvimento de produtos de software como uma seqüência de atividades com algum resultado de uma atividade para outra, o Modelo Espiral é representado como uma espiral. Um Modelo Espiral é dividido em um certo número de atividades que formam uma estrutura chamada de regiões de tarefas. Tipicamente, há de três a seis regiões de tarefas. A Figura 2.4 mostra o modelo espiral com seis regiões de tarefas:

- **Comunicação com o cliente**. Tarefas necessárias para estabelecer efetiva comunicação entre o engenheiro de software e o cliente;
- **Planejamento**. Tarefas necessárias para definir recursos, prazos e outras informações relacionadas ao projeto;
- **Análise de risco**. Tarefas necessárias para avaliar os riscos, tanto técnicos quanto gerenciais;
- **Engenharia**. Tarefas necessárias para construir um ou mais representações do produto de software;

- **Construção e liberação**. Tarefas necessárias para construir, testar, instalar e fornecer apoio ao usuário (p. ex., documentação e treinamento);
- **Avaliação pelo cliente**. Tarefas necessárias para obter *feedback* do cliente, com base na avaliação das representações do produto de software criadas durante o estágio de engenharia e implementadas durante o estágio de instalação.

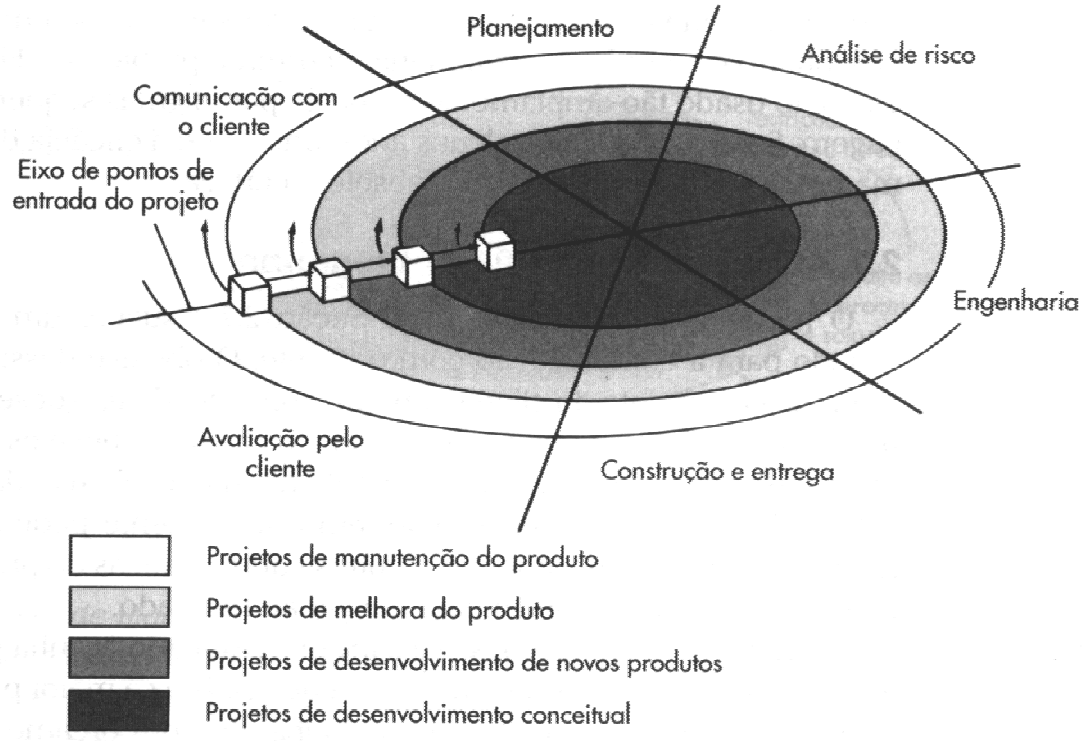


Figura 2.4 – Modelo Espiral (Fonte: PRESSMAN, 2002)

Assim que esse processo evolucionário tem início, a equipe de engenharia de software move-se em volta da espiral, no sentido horário, a partir do centro. O primeiro circuito em torno da espiral pode resultar no desenvolvimento da especificação do produto; passagens subseqüentes em torno da espiral podem ser usadas para desenvolver um protótipo e, depois, progressivamente, versões mais sofisticadas do produto de software. Cada passagem pela região de planejamento resulta em ajustes ao plano do projeto. O custo e o cronograma são ajustados com base no *feedback* da avaliação do cliente. Além disso, o gerente de projeto ajusta a quantidade de iterações necessárias planejada para completar o produto de software (PRESSMAN, 2002).

O Modelo Espiral é uma abordagem realista para o desenvolvimento de sistemas de grande porte, pois exige a consideração direta dos riscos técnicos em todos os estágios do

projeto e, se aplicado adequadamente, deve reduzir os riscos antes que eles fiquem problemáticos.

Mas, como outros paradigmas, o Modelo Espiral não é uma panacéia (PRESSMAN, 2002). Pode ser difícil convencer os clientes (particularmente em situações de contrato), que essa abordagem é controlável. Exige competência considerável na avaliação de risco e depende dessa competência para ter sucesso. Se um risco importante não for descoberto e gerenciado, fatalmente ocorrerão problemas. Finalmente, o Modelo Espiral não tem sido usado tão amplamente como o Modelo Cascata ou o Modelo Prototipagem. Será necessário algum tempo, antes que a eficácia desse importante paradigma possa ser determinada com absoluta certeza.

2.3.4 Modelo de Desenvolvimento Baseado em Componentes

O Modelo de Desenvolvimento Baseado em Componentes (Figura 2.5) incorpora muitas características do Modelo Espiral. Evolucionário por natureza (NIERSTRASZ *et al*, 1992), demanda uma abordagem iterativa para a criação do produto de software. Todavia, o Modelo Baseado em Componentes compõe aplicações a partir de componentes de produtos de software previamente preparados (chamados classes).

Classes criadas em projetos anteriores de engenharia de software são armazenadas em uma biblioteca ou repositório de classes. Uma vez identificadas as classes adequadas, a biblioteca de classes é examinada para determinar se elas existem ou se são similares àquelas necessárias. Em caso afirmativo, são extraídas da biblioteca e reusadas. Se uma classe adequada não é encontrada na biblioteca, passa por engenharia usando métodos orientados a objetos. Então, é composta a primeira iteração da aplicação a ser construída, usando classes extraídas da biblioteca e quaisquer novas classes construídas para satisfazer as necessidades específicas da aplicação. O fluxo de processo volta para o Modelo Espiral e, em última análise, reingressa na iteração de montagem de componentes durante as passagens subseqüentes pela atividade de engenharia.

Segundo SOMMERVILLE (2003), o Modelo Baseado em Componentes tem a vantagem de reduzir a quantidade de produtos de software a ser desenvolvida e, assim, de reduzir custos e riscos. Geralmente, ele também propicia a entrega mais rápida do produto de software. Contudo, as adequações nos requisitos são inevitáveis e isso pode resultar em um produto de software que não atenda às reais necessidades dos usuários. Além disso,

algum controle sobre a evolução do produto de software pode se perder, uma vez que novas versões dos componentes reutilizáveis não estão sob o controle da organização que utiliza esses componentes.

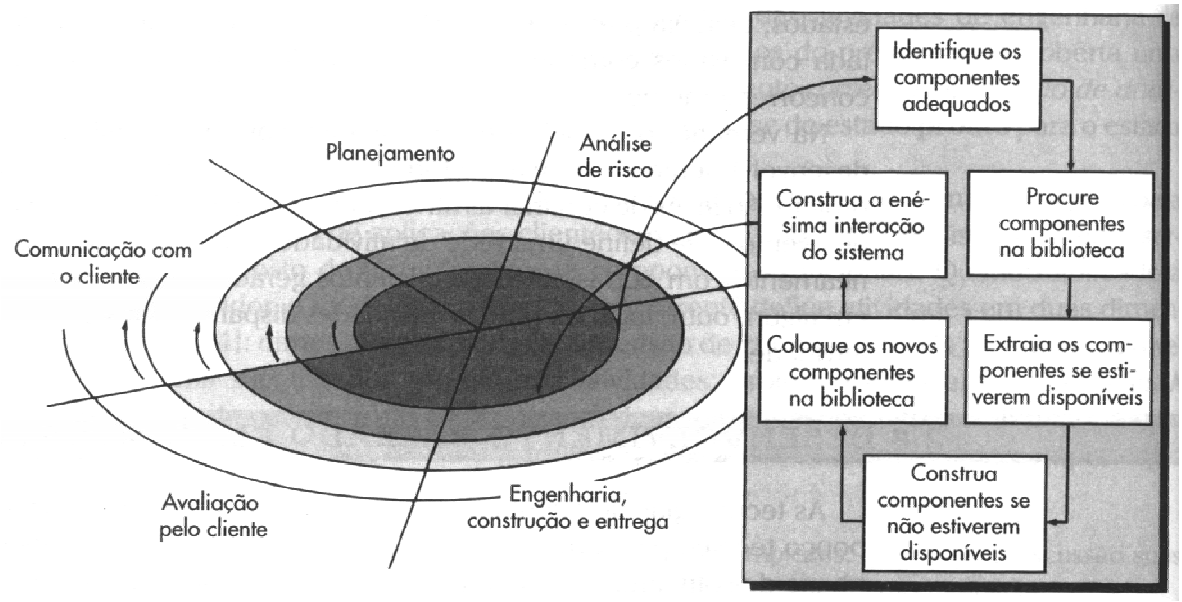


Figura 2.5 – Modelo de Desenvolvimento Baseado em Componentes (Fonte: PRESSMAN, 2002)

2.3.5 Modelo de Métodos Formais

O Modelo de Métodos Formais (Figura 2.6) consiste de um conjunto de atividades que levam à especificação matemática formal do produto de software. Métodos formais permitem ao engenheiro de software especificar, desenvolver e verificar um produto de software pela aplicação de uma rigorosa notação matemática (PRESSMAN, 2002).

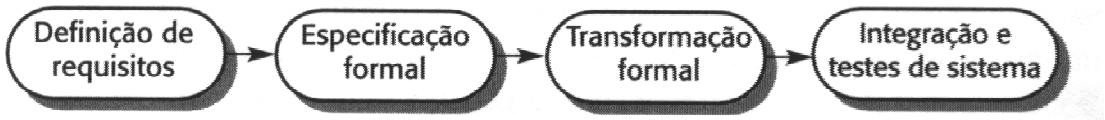


Figura 2.6 – Modelo de Métodos Formais (Fonte: SOMMERVILLE, 2003)

De acordo com SOMMERVILLE (2003), o desenvolvimento formal de produtos de software é uma abordagem de desenvolvimento que tem algo em comum com o Modelo Cascata, possuindo as seguintes distinções:

- A especificação de requisitos do produto de software é redefinida em uma especificação formal detalhada, que é expressa em uma notação matemática;
- O projeto, a implementação e o teste de unidade são substituídos por um processo de desenvolvimento transformacional, em que a especificação formal é refinada por meio de uma série de transformações.

No processo de transformação, a representação matemática formal do sistema é sistematicamente convertida em uma representação de sistema mais detalhada, mas ainda matematicamente correta. Cada etapa acrescenta mais detalhes, até que a especificação formal seja convertida em um produto de software equivalente. As transformações são suficientemente próximas, de modo que o esforço de verificação da transformação não é excessivo. Portanto, pode-se garantir, considerando que não haja erros de verificação, que o produto de software é uma verdadeira implementação da especificação.

Esse modelo tem sido aplicado ao desenvolvimento de sistemas críticos, especialmente naqueles onde a segurança é um fator crítico (ex: sistema de controle de tráfego aéreo). No entanto, a necessidade de habilitações especializadas para aplicar as técnicas de transformação e a dificuldade de especificar formalmente alguns aspectos do sistema, tais como a interface com o usuário, são fatores que limitam seu uso.

2.3.6 Técnicas de Quarta Geração

O termo técnicas de quarta geração (*fourth generation techniques*, 4GT) abrange um amplo conjunto de ferramentas de produtos de software que têm uma coisa em comum: cada ferramenta permite ao engenheiro de software especificar alguma característica do produto de software em alto nível. A ferramenta gera código-fonte automaticamente baseado na especificação do engenheiro de software. Há um consenso de que quanto mais alto o nível de especificação do produto de software para uma máquina, mais rapidamente ele pode ser construído. O paradigma 4GT para engenharia de software concentra-se na habilidade de especificar produtos de software usando formas especializadas de linguagem ou uma anotação gráfica que descreve o problema a ser resolvido de forma que o cliente possa entender.

Para transformar uma implementação 4GT em um produto, o engenheiro de software precisa realizar testes exaustivos, desenvolver documentação adequada e realizar todas as outras atividades de integração da solução, necessárias em outros paradigmas de engenharia de software. Além disso, o produto de software desenvolvido por 4GT deve ser construído de uma forma que permita a realização rápida da manutenção.

Como todos os paradigmas de engenharia de software, o modelo 4GT tem vantagens e desvantagens. Os proponentes reivindicam, para o pessoal que constrói o produto de software, uma drástica redução no tempo de desenvolvimento e uma melhora significativa

na produtividade. Os oponentes alegam que as atuais ferramentas 4GT não são tão mais fáceis de usar do que as linguagens de programação; que o código-fonte resultante, produzido por tais ferramentas, é ineficiente; e a manutenibilidade de grandes produtos de software desenvolvidos usando 4GT está aberta a discussão.

Há algum mérito nas alegações de ambas as partes e o estado atual das abordagens pode ser resumido em (PRESSMAN, 2002):

- O uso de 4GT é uma abordagem viável para muitas áreas de aplicação diferentes. Combinadas com ferramentas de engenharia de software apoiadas por computador e geradores de código, as 4GT oferecem uma solução adequada a muitos problemas dos produtos de software;
- Dados coletados por empresas que usam 4GT indicam que o tempo necessário para desenvolver produtos de software é bastante reduzido em pequenas e médias aplicações, e que em grandes aplicações a quantidade de projeto e análise também é reduzida;
- Todavia, o uso de 4GT para grandes esforços de desenvolvimento de produtos de software exige tanto ou mais análise, projeto e teste, para alcançar a substancial redução de tempo resultante da eliminação da codificação.

Em resumo, as técnicas de quarta geração se tornaram uma parte importante da engenharia de software. Quando combinado com as abordagens de desenvolvimento baseado em componentes, o paradigma 4GT pode se transformar na abordagem predominante para o desenvolvimento de produtos de software.

2.3.7 Processo Unificado

O Processo Unificado (*Unified Process* – UP) é o conjunto de atividades necessárias para transformar requisitos do usuário em um produto de software (JACOBSON *et al*, 1999). O UP utiliza a Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML) no preparo de todos os artefatos do sistema (JACOBSON, 1998) (JACOBSON *et al*, 1999). Os aspectos que distinguem o processo unificado são capturados em três conceitos chave (JACOBSON, 1998) (JACOBSON *et al*, 1999):

- Direcionado a Casos de Uso: um caso de uso (JACOBSON, 1998) (JACOBSON *et al*, 1999) (ERIKSSON, 1998) é um pedaço de funcionalidade do sistema que dá ao usuário um resultado de valor. Casos de uso capturam requisitos funcionais e todos juntos resultam no modelo de casos de uso, o qual descreve a funcionalidade completa do

sistema. Com base no modelo de casos de uso, são criados o Modelo de Análise, o Modelo de Projeto e o Modelo de Implementação que realizam estes casos de uso;

- Centrado na Arquitetura: o conceito de arquitetura de software incorpora os aspectos estáticos e dinâmicos mais importantes do sistema. A arquitetura é influenciada por muitos fatores, tais como a plataforma de software sobre a qual o sistema vai rodar (sistema operacional, sistema gerenciador de banco de dados, protocolos para comunicação em rede), blocos de construção reusáveis disponíveis (por exemplo, um *framework* para construção de interface gráfica com o usuário), considerações de distribuição, sistemas legados e requisitos não funcionais (performance, confiabilidade). Ela representa uma visão do projeto como um todo, na qual as características mais importantes são colocadas em destaque, deixando os detalhes de lado. O Processo Unificado é centrado em arquitetura, pois defende a definição de um esqueleto para a aplicação (a arquitetura), a ganhar corpo gradualmente ao longo do desenvolvimento;
- Iterativo e Incremental: É mais prático dividir o trabalho em pedaços menores ou miniprojetos. Cada miniprojeto é uma iteração que resulta em um incremento. Iterações são passos em um fluxo de trabalho e incrementos são crescimentos do produto. O UP é iterativo e incremental, oferecendo uma abordagem para particionar o trabalho em porções menores ou mini-projetos.

O ciclo de vida adotado no UP é tipicamente evolutivo. Contudo, uma forma de organização em fases é adotada para comportar os ciclos de desenvolvimento, permitindo uma gerência mais efetiva de projetos complexos. Ao contrário do tradicionalmente definido como fases na maioria dos modelos de ciclo de vida – planejamento, levantamento de requisitos, análise, projeto, implementação e testes, são definidas fases ortogonais a estas, a saber (JACOBSON *et al*, 1999):

- Concepção: geração de um modelo de casos de uso simplificado, que contenha os casos de uso mais críticos. Nesta fase, a arquitetura é experimental, apenas um esboço contendo os subsistemas mais cruciais;
- Elaboração: a maioria dos casos de uso do produto é especificada em detalhes e a arquitetura do sistema é projetada;
- Construção: um produto completo (modelos, diagramas) é desenvolvido de maneira iterativa e incremental, para que esteja pronto para a transição aos usuários. Embora a

arquitetura do sistema encontre-se estável, os desenvolvedores podem descobrir maneiras melhores de estruturá-lo e podem sugerir pequenas mudanças aos arquitetos;

- Transição: esta fase cobre o período durante o qual o produto fica em versão beta. Nessa versão, um pequeno número de usuários experientes utiliza o produto e relata defeitos e deficiências. Desenvolvedores os corrigem e incorporam algumas das melhorias sugeridas. Esta fase envolve atividades como, treinamento de clientes, fornecimento de assistência *on-line* e correção de defeitos encontrados depois da distribuição. A equipe de manutenção freqüentemente divide os defeitos em duas categorias: aqueles com efeitos operacionais que justificam correção imediata e aqueles que podem ser corrigidos na próxima versão regular.

A ênfase principal de cada uma destas fases são apresentadas na Figura 2.7.

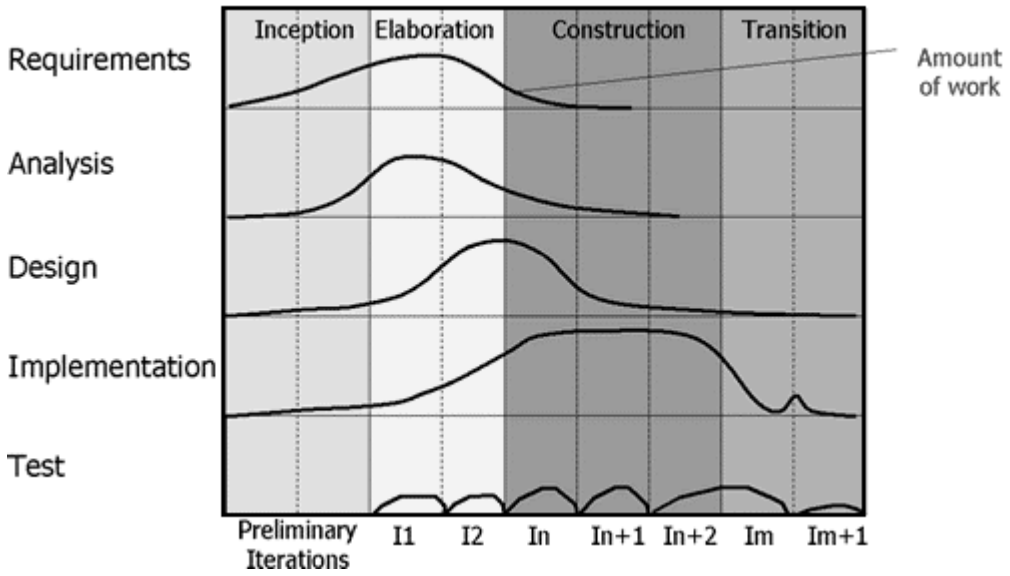


Figura 2.7 – A ênfase principal de cada uma das fases do UP (Fonte: HEPTAGON, 2005)

Além dos conjuntos de iterações em cada fase, as fases em si podem ocorrer de forma iterativa. Como mostra a Figura 2.8, elas não necessariamente têm a mesma duração. O esforço realizado em cada uma varia fortemente em função das circunstâncias específicas do projeto.

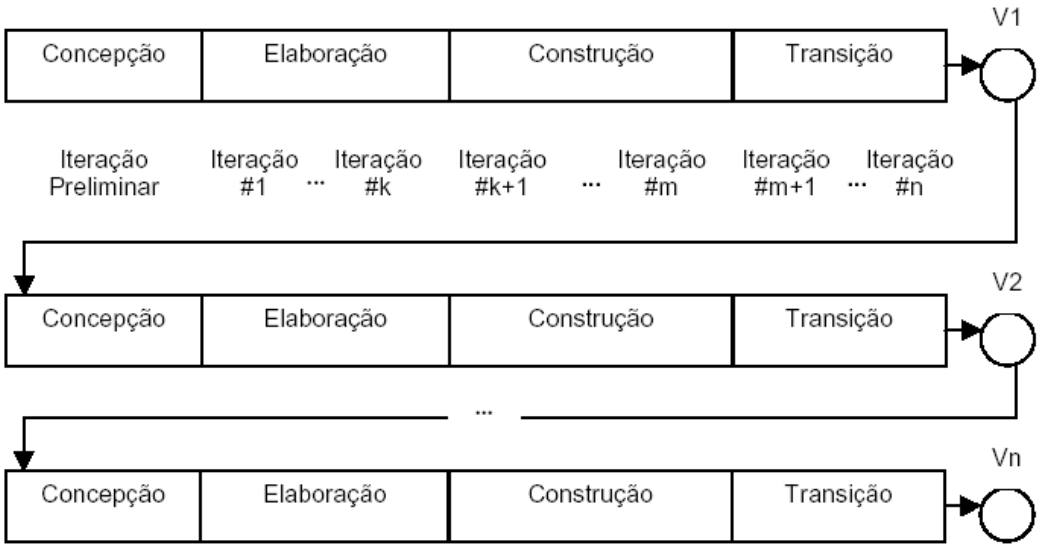


Figura 2.8 – O Modelo de Ciclo de Vida proposto no UP (Fonte: HEPTAGON, 2005)

2.4 Considerações Finais

Neste capítulo, foi destacada a importância do processo de desenvolvimento de produtos de software. Em seguida, foram apresentados alguns modelos de processo de desenvolvimento, cada qual exibindo pontos fortes e fracos, mas todos tendo uma série de fases genéricas em comum.

É extremamente importante selecionar o processo de desenvolvimento adequado para a construção de produtos de software, pois só assim haverá garantia de qualidade, confiabilidade e segurança.

3 MANUTENÇÃO DE PRODUTOS DE SOFTWARE

3.1 Considerações Iniciais

A manutenção de produtos de software é a atividade durante a qual ocorrem modificações em um ou mais artefatos de software resultantes do desenvolvimento de um produto de software (ou de manutenções anteriores), buscando mantê-lo disponível, corrigir suas falhas, melhorar seu desempenho e adequá-lo aos requisitos novos ou modificados, conforme as necessidades de seus usuários (ANSI/IEEE, 1983).

Vários são os problemas técnicos e gerenciais enfrentados pela atividade de manutenção de produtos de software, cujas principais causas são (BOOCH, 1994): a complexidade do domínio do problema, a dificuldade em gerenciar o processo de desenvolvimento e manutenção, a eventual necessidade de flexibilização do produto de software e a tecnologia ultrapassada presente nos sistemas legados, os quais residem de maneira significativa a modificações ou evoluções. Estes sistemas legados têm, em média, entre 10 e 20 anos de idade (PRESSMAN, 2001), geralmente os atuais mantenedores não são os mesmos que trabalharam na sua criação, pouca ou nenhuma metodologia de projeto foi aplicada e, por conseguinte, o resultado é um duvidoso projeto arquitetural (e de dados). A documentação destes sistemas é incompleta e o registro das mudanças superficial.

Este capítulo aborda a importância, as dificuldades, as tecnologias utilizadas e alguns modelos de processo de manutenção de produtos de software.

3.2 Importância da Manutenção

Uma ampla pesquisa realizada por LIENTZ & SWANSON (1980) no final dos anos 70, e repetida por outros, em diferentes domínios, expôs a alta fração do custo do ciclo de vida que estão sendo gastos em manutenção. LIENTZ & SWANSON (1980) categorizaram a manutenção em quatro classes:

- Adaptativa: realizada quando o produto de software precisa ser adaptado às novas tecnologias (hardware e software) implantadas no ambiente operacional;
- Evolutiva: realizada quando o produto de software deve englobar novos requisitos ou melhorias decorrentes da evolução na tecnologia de implementação empregada;

- Corretiva: realizada quando são corrigidos erros não identificados durante a fase de testes;
- Preventiva: realizada quando o produto de software é alterado para aumentar sua manutenibilidade e/ou confiabilidade. Este tipo de manutenção é relativamente raro em ambientes de desenvolvimento. Modificações realizadas neste tipo de manutenção não afetam o comportamento funcional do produto de software

A pesquisa demonstrou que em torno de 75% dos esforços de manutenção era dos dois primeiros tipos e correção de erros consumia a 21%. Conforme abordado em DIAS *et al* (2003), a demanda do mercado por mantenedores de produtos de software é alta por ser uma atividade que consome a maior parte dos custos da indústria do software. PFLEEGER (2001) estima esses custos em cerca de 80% do orçamento total do ciclo de vida de um produto de software. Muitos estudos subseqüentes demonstraram uma magnitude similar do problema. Esses estudos demonstraram que a incorporação de novas necessidades dos usuários é o problema central da evolução e manutenção de produtos de software.

Segundo BENNETT & RAJLICH (2000), se as mudanças puderem ser antecipadas na fase de projeto, elas podem ser construídas sob alguma forma de parametrização. O problema fundamental apoiado por 40 anos de uma vasta experiência é a quantidade de mudanças necessárias que os próprios desenvolvedores não imaginaram. Então, há a necessidade em desenvolver de produtos de software fáceis de serem mantidos, pois: i) a manutenção consume uma grande parte de todo o custo gasto no ciclo de vida do software; e ii) a falta de habilidade em mudar produtos de software rapidamente e de maneira confiável implica na perda de oportunidades do negócio. Esses são problemas ainda não resolvidos, implicando que pesquisas sobre manutenção tendem a aumentar nos próximos anos.

3.3 Dificuldades da Manutenção

Pode-se dizer que a manutenção de software é um processo de mudança para garantir a sobrevivência do produto de software. No entanto, segundo SCHNEIDEWIND (1987), o processo de manutenção é bastante árduo devido aos seguintes fatores:

- Não existem formas ainda eficazes de acompanhar os produtos de software gerados e o processo de sua geração na manutenção. Os modelos disponíveis para garantia da

qualidade de produtos de software são bastante completos, porém muito focados no desenvolvimento de novos projetos;

- Dependendo de como o produto de software foi construído, a realização de mudanças em geral desencadeia um efeito dominó sobre o qual tem-se muita dificuldade de previsão. Isso se deve ao fato de não ter uma ferramenta apropriada para identificação de conseqüências que uma manutenção pode acarretar. Este efeito é citado também por RAJLICII & SRIDIIAR (1996);
- Em geral, a manutenção é apresentada na área de Engenharia de Software como um processo iniciado imediatamente após a implantação do produto de software. Isso não seria um problema, não fosse o pouco ou nenhum envolvimento das equipes de manutenção durante o processo de desenvolvimento.

COUSIN & COLLOFELO (1992) acreditam que as equipes de manutenção poderiam resolver melhor todos os problemas de manutenção se dispusessem de documentação atualizada, treinamento contínuo e ferramentas automatizadas. Uma combinação destes três itens poderia ser um bom caminho para a continuidade da qualidade do software.

Focando mais diretamente o processo de documentação do produto de software, esforços têm sido feitos na tentativa de melhorar tal processo. Segundo WONG *et al* (1995), mais de 50% do trabalho de evolução do produto de software é dedicado ao entendimento do programa. A tarefa mais difícil da manutenção de produtos de software é o entendimento da sua estrutura (TILLEY, 1993). No estudo de GARLAND & CALLISS (1991), foi discutido um método para incorporar informações sobre a solução de problemas em uma base de dados que contém todo o acompanhamento da manutenção.

Muitas das questões citadas acima levam a identificar que a falta de modelos de qualidade definidos especificamente para a manutenção ou adaptados para esta atividade, assim como a carência de ferramentas que permitam a implementação destes modelos, pode vir a deteriorar a qualidade do produto de software ao longo do processo de manutenção.

3.4 Técnicas de Manutenção

No processo de manutenção, quando se trata de reconstruir um produto de software, ou seja, realizar sua reengenharia, basicamente, a informação do projeto é extraída do

código-fonte, para fornecer uma visão geral do objetivo do produto de software ou para reconstruir parte ou todo o sistema utilizando tecnologia moderna (CAPRETZ & CAPRETZ, 1996).

Infelizmente, a reengenharia necessita de uma terminologia padrão e, para tal, diferentes autores geralmente utilizam diferentes termos para explicar o mesmo conceito. Tentativas foram feitas com o propósito de clarear termos como engenharia reversa, redocumentação, recuperação de projeto e reestruturação, os quais são descritos a seguir.

Engenharia reversa é o processo de analisar um produto de software com o objetivo de identificar seus componentes e seus relacionamentos, criando representações de outra forma ou em um nível de abstração mais alto com o intuito de gerenciar sua complexidade (CAPRETZ & CAPRETZ, 1996).

Redocumentação, como uma subárea da engenharia reversa, objetiva recuperar a documentação de um sistema, através da criação ou revisão de uma representação semanticamente equivalente, dentro do mesmo nível relativo de abstração, sendo que as formas resultantes de representação são consideradas como visões alternativas, utilizadas para uma melhor compreensão humana do produto de software analisado (CHIKOFSKY & CROSS, 1990).

Recuperação de Projeto é uma subárea da engenharia reversa, na qual o conhecimento do domínio da aplicação, as informações externas e a dedução são adicionados às observações referentes ao produto de software, para extraírem abstrações significativas de mais alto nível, além daquelas obtidas através da observação direta (CHIKOFSKY & CROSS, 1990). Ela deve reproduzir a informação necessária para que uma pessoa esteja apta a entender plenamente o programa; especificamente, o que o programa faz, como e por que ele faz isso (CAPRETZ & CAPRETZ, 1996).

Reestruturação é a transformação de uma forma de representação, para outra no mesmo nível de abstração relativo, preservando o comportamento externo do produto de software (funcionalidade e semântica). Geralmente usada como uma forma de manutenção preventiva, a reestruturação é aplicada em produtos de software que tenham sido desenvolvidos de forma desestruturada, resultando uma representação que preserva as suas características, porém de forma mais bem estruturada (CAPRETZ & CAPRETZ, 1996).

3.5 Modelos de Processo de Manutenção

De acordo com CAPRETZ & CAPRETZ (1996), a manutenção, vista como uma simples fase do ciclo de vida de um produto de software, deve ser substituída por um modelo refletindo os aspectos da evolução. É necessária uma nova e mais abrangente perspectiva do ciclo de vida do software, enfatizando mudança, manutenção e migração para novas tecnologias.

Um modelo de manutenção de produto de software descreve os passos necessários para satisfazer uma necessidade de manutenção. Diferentes tipos de necessidades, características de ambientes e limites orçamentários exigem diferentes modelos.

Um dos problemas relativos à manutenção é a falta de bons modelos. Os modelos de manutenção tendem a ser improvisados e executados sem o acesso à informação necessária. Além disso, a tecnologia para manutenção (por exemplo, métodos para leitura de projeto e de código e ferramentas automatizadas) é inferior à existente para o desenvolvimento de produtos de software. O resultado é uma manutenção improdutiva e suscetível a erros (CAPRETZ & CAPRETZ, 1996).

Vários autores propuseram modelos de manutenção de produtos de software. Os modelos apresentados sucintamente nesta seção foram propostos entre os anos 70 e 90 e foram pouco modificados de modo a acompanhar a evolução das tecnologias emergentes. Segundo CAPRETZ & CAPRETZ (1996), os modelos foram enquadrados em três fases distintas apresentando sua evolução, refletindo a ordem cronológica em que foram desenvolvidos. Os primeiros modelos são de alto nível, pois exibem uma visão geral, não entrando em detalhes do processo de manutenção. Os mais recentes introduzem princípios avançados da Engenharia de Software.

3.5.1 Modelos de Manutenção de Software Antigos

Os primeiros modelos de manutenção de produtos de software foram bem simples. Eles provêem a ordem de fases a serem seguidas, omitindo detalhes de como devem ser executadas. São eles:

- **Modelo de Boehm**: (BOEHM, 1976) destaca apenas três fases do modelo de manutenção: i) entender o produto de software existente; ii) modificar o produto de software existente; e iii) revalidar o produto de software existente.

- **Modelo de Yau & Collofello**: (YAU & COLLOFELLO, 1980) identificam quatro fases básicas de modo a gerenciar um processo de manutenção de produtos de software. O modelo é focado na estabilidade do sistema através da análise de efeitos colaterais causados por mudanças. As fases são: i) entender o produto de software (consiste em analisá-lo e entendê-lo); ii) gerar uma sugestão de manutenção (consiste em gerar uma proposta para realizar a implementação do objetivo da manutenção; isso requer um pleno entendimento dos objetivos e do produto de software a ser modificado); iii) descrever os efeitos colaterais (consiste em descrever todos os efeitos colaterais relativos à mudança no produto de software); e iv) teste (consiste em testar o produto de software com o intuito de verificar se ele possui a mesma confiabilidade existente no antigo produto de software). Segundo CAPRETZ & CAPRETZ (1996), cada uma destas quatro fases é conectada aos atributos da qualidade do produto de software. A primeira está associada à complexidade, documentação e autodescrição dos atributos, o que contribui para um melhor entendimento do programa. Na segunda fase, a forma como será gerada a sugestão de manutenção é afetada pelo atributo compreensão. YAU & COLLOFELLO (1980) dizem que uma das fases mais importantes relativos ao atributo qualidade é a estabilidade do programa (fase três), porque se a estabilidade for precária, o impacto de qualquer mudança é enorme. Na quarta fase, o fator primário que contribui para as técnicas de teste de custo-benefício do desenvolvimento é a validação do produto de software.

3.5.2 Modelos de Processo de Manutenção Intermediários

Os modelos descritos nesta subseção são mais bem elaborados do que os da subseção anterior. O Modelo de Martin & Macclure (MARTIN & MACCLURE, 1983) descreve detalhes de como executar a manutenção; enquanto o Modelo de Arthur (ARTHUR, 1988) é mais compreensivo, fornecendo guias para realizar a manutenção de acordo com as necessidades do sistema.

- **Modelo de Martin & McClure**: conforme MARTIN & MACCLURE (1983), as três fases básicas da manutenção são descritas e detalhadas a seguir:
    - ► **Entender o programa**. Exige um entendimento do objetivo funcional, da estrutura interna e das necessidades operacionais de um produto de software. Essa função é subdividida em três subfunções:

1. **Entendimento Cima-Embaixo**. Um entendimento geral é necessário para se tornar familiar com o produto de software, juntamente com um entendimento detalhado;
2. **Melhorar a Documentação**. Durante o entendimento do produto de software, documentar o que foi aprendido, concentrando no melhoramento da documentação de alto nível;
3. **Participação no Desenvolvimento**. Quando possível, participar do processo de desenvolvimento para aprender sobre o produto de software.

- **Modificar o programa**. Envolve a criação de uma nova lógica para corrigir um erro ou implementar uma mudança Os seguintes passos são necessários:
    1. **Planejar a mudança no produto de software**. Uma estratégia é necessária para revisar o produto de software. Inicialmente, os módulos e as estruturas de dados a serem alterados são isolados. A seguir, eles são estudados detalhadamente. As mudanças são projetadas, especificando a nova lógica e a lógica existente a ser alterada;
    2. **Alterar o código do produto de software de modo a incorporar a mudança**. Os objetivos deste passo são: i) codificar eficientemente e corretamente a mudança; e ii) eliminar qualquer efeito indesejado decorrente da mudança.
- **Revalidar o programa**. Os mantenedores deverão testar o produto de software para demonstrar que a nova lógica está correta e a porção do produto de software que não foi modificada permanece intacta. Os mantenedores deverão:
    1. Realizar testes de produto de software com o intuito de falhas serem identificadas, assegurando que o produto de software, de uma forma geral, está funcionando corretamente;
    2. Testar as partes não modificadas do produto de software, executando testes de regressão para determinar se ainda funcionam corretamente;
    3. Testar as partes modificadas do produto de software, para determinar se as mudanças foram projetadas e implementadas corretamente.

- **Modelo de Arthur**: ARTHUR (1988) propôs um modelo mais elaborado e compreensivo para manutenção de produtos de software. As fases de seu modelo lidam com as necessidades de mudanças até que sejam implementadas, testadas e liberadas para os usuários. As sete fases que descrevem este modelo são:

- **Gerenciar as mudanças**. O objetivo básico é identificar, descrever e traçar o status de cada necessidade. Nesta fase, as mudanças necessárias são geradas e analisadas;
- **Analisar as mudanças**. O objetivo principal é determinar o escopo da mudança necessária como uma base para o planejamento e implementação. As mudanças necessárias são analisadas, identificando o possível impacto nos produtos de software existentes, documentação, hardware, estrutura de dados e humanos (usuários, mantenedores e operadores);
- **Planejar liberações do produto de software**. O principal objetivo é planejar o tempo para liberação do produto de software ou parte dele;
- **Projetar as mudanças**. Um projeto lógico, para relatar o nível do produto de software, e um físico, para relatar o nível do programa, são desenvolvidos para as mudanças aprovadas;
- **Mudanças no código**. O objetivo é mudar o produto de software de modo a refletir as mudanças aprovadas e representadas nos projetos lógico e físico;
- **Testes**. O objetivo primário é assegurar conformidade com as mudanças originais e as aprovadas;
- **Liberar o produto de software**. Entregar o produto de software e a documentação atualizada aos usuários para instalação e execução.

3.5.3 Modelos de Processo de Manutenção Recentes

Os modelos presentes nesta seção são o Modelo de Foster (FOSTER *et al*, 1989) e o Modelo Dirigido a Necessidades (BENNETT *et al*, 1991). Enquanto o primeiro modelo apresenta a manutenção de um ponto de vista organizacional, o segundo modelo descreve as atividades de manutenção de um produto de software como sendo mudanças ordenadas por usuários, envolvendo um estrito controle de gerenciamento.

- **Modelo de Foster**: FOSTER *et al* (1989) debatem que a organização da manutenção é o fato mais importante para o seu sucesso. Seu modelo é derivado de observações feitas por equipes de mantenedores abrangendo questões técnicas e gerenciais. Esse modelo difere dos outros por não citar fases a serem seguidas, mas por referir às funções executadas pelo pessoal envolvido. O modelo especifica as seguintes funções as quais envolvem perguntas, relatórios referentes a problemas e mudanças requeridas por usuários.

- **Mesário frontal**: comunica com os usuários obtendo as necessidades e as armazenando em forma de documentos. É sua responsabilidade fornecer respostas/soluções direcionadas ou transmiti-las para uma equipe mais especializada;
- **Arquivo de pedidos**: recebe esses documentos, os quais demandam novas soluções. Cada documento é enfileirado até que se torne viável a execução da solução referente a ele. Essa fila é representada como um arquivo de pedidos que não foram solucionados. O gerenciamento deste arquivo é uma função muito importante, pois prioridades são assimiladas e o futuro é planejado através de investigações preliminares e análises de impacto. Se as investigações preliminares revelarem que a equipe não possui recursos ou capacidade para resolver um problema, o pedido é transferido para outra equipe, que a solucionará. Se a equipe for capaz de resolver o problema, as necessidades de maiores prioridades são retiradas do arquivo e a mudança do produto de software é projetada;
- **Arquivo de mudanças**: acumula as mudanças e soluções recebidas de outras equipes. De tempos em tempos, uma decisão é tomada, de modo a desenvolver uma nova versão do produto de software, incorporando todas as mudanças viáveis;
- **Arquivo de soluções**: fornece um conjunto de conhecidas respostas/soluções, as quais são acessadas pelo mesário frontal. A informação é representada em várias formas, como versões de produtos de software e documentos com respostas a perguntas freqüentes. Se a solução no arquivo pode rapidamente ser gerada, ela é imediatamente resolvida e repassada ao usuário. Novas soluções são também armazenadas neste arquivo, onde são avaliadas para distribuição aos usuários originais.

- **Modelo Dirigido a Necessidades**: descreve as atividades de manutenção de produtos de software como se as necessidades de mudanças fossem ordens de usuários. O modelo consiste em seguir três grandes processos, os quais envolvem um estrito controle de gerenciamento: i) controle de necessidades; ii) controle de mudanças; e iii) controle de liberações. O controle de pedidos lida com necessidades de usuários. As principais atividades desenvolvidas nesta fase são: i) coleção de informações sobre cada necessidade; ii) selecionar mecanismos para categorizar tais necessidades; iii) uso de

análise de impacto para verificar o custo e o benefício de cada necessidade; e iv) definição de prioridades para cada necessidade. O controle de mudanças é geralmente visto como o processo chave, por analisar o código existente, o que é uma atividade cara. As atividades envolvidas neste processo são: i) seleção de mudanças de acordo com suas prioridades; ii) reprodução do problema (se existir); iii) análise do código, documentação e especificação; iv) projeto de mudanças e testes; e iv) assegurar a qualidade. É durante o controle de liberações que as necessidades a serem incluídas na nova versão do produto de software são decididas e as mudanças necessárias no código são realizadas. Neste processo, as seguintes atividades são realizadas: i) determinação da versão; ii) desenvolvimento da nova versão através de edição do código e gerenciamento de arquivos e configurações, assegurando a qualidade; iii) teste de confidência; iv) distribuição; e v) teste de aceitação.

3.6 Considerações Finais

O presente capítulo apresentou uma visão geral da manutenção de produtos de software, abordando algumas técnicas, algumas dificuldades e alguns modelos.

A manutenção de produtos de software é uma atividade complexa e, embora exista um número relativamente grande de modelos, estes têm mostrado falhas que requerem pesquisas adicionais neste campo, o que justifica iniciativas de estudos e pesquisas referentes a essa área.

4 UML – *Unified Modeling Language*

4.1 Considerações Iniciais

A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de software (FURLAN, 1998). A UML prepara uma forma-padrão para a preparação de planos de arquitetura de projetos de produtos de software, incluindo aspectos conceituais tais como processos de negócios e funções do produto de software, além de itens concretos como as classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de software reutilizáveis (BOOCH *et al*, 1999).

Uma breve história da linguagem, a importância da modelagem, uma visão geral e uma descrição sucinta dos diagramas propostos pela UML são apresentadas neste capítulo.

4.2 Importância da Modelagem

Para que uma empresa de software seja bem-sucedida é necessário fornecer produtos de software de qualidade e ser capaz de atender às necessidades dos respectivos usuários. Uma empresa, que consiga desenvolver esse produto de software de maneira previsível e em determinado período, com utilização eficiente e eficaz de recursos, será uma empresa com um negócio viável.

Entretanto, para entregar um produto de software que satisfaça ao propósito pretendido e desenvolvê-lo com qualidade duradoura e de forma rápida, eficiente e efetiva, com o mínimo de desperdício e de retrabalho, é preciso reunir e interagir com os usuários de uma maneira disciplinada, com a finalidade de expor os requisitos reais do produto de software, criar uma arquitetura de fundação sólida que aceite modificações e dispor das pessoas certas, das ferramentas adequadas e do enfoque corretos. Para fazer isso de maneira previsível e, consistente, com uma avaliação dos custos reais do produto de software, é preciso utilizar um processo seguro de desenvolvimento que possa ser adaptado às novas necessidades de seu negócio e de sua tecnologia.

A modelagem é uma parte central de todas as atividades que levam à implantação de um bom produto de software. Modelos são construídos para comunicar a estrutura e o comportamento desejados, visualizar e controlar a arquitetura e compreender melhor o produto de software que está sendo elaborado, muitas vezes expondo oportunidades de

simplificação e reaproveitamento. Além disso, os modelos facilitam a gerência de riscos do projeto.

BOOCH *et al* (1999) define modelo como sendo uma simplificação da realidade. Os modelos fornecem uma cópia do projeto de um produto de software que poderão abranger planos detalhados, assim como planos mais gerais com uma visão panorâmica. Um bom modelo inclui componentes que têm ampla repercussão e omite os componentes menores que não são relevantes em determinado nível de abstração. Os produtos de software podem ser descritos sob diferentes aspectos, com a utilização de modelos distintos e cada modelo será uma abstração semanticamente específica. Os modelos podem ser estruturais, dando ênfase à organização, ou podem ser comportamentais, dando ênfase à dinâmica.

Com a modelagem, são alcançados quatro objetivos (BOOCH *et al*, 1999): i) os modelos ajudam a visualizar o produto de software como ele é ou como é desejado ser; ii) os modelos permitem especificar a estrutura ou o comportamento do produto de software; iii) os modelos proporcionam um guia para a construção do produto de software; e iv) os modelos documentam as decisões tomadas.

4.3 História da UML

Os primeiros precursores das linguagens de modelagem orientadas a objetos surgiram no meio da década de 70 e continuaram a aparecer durante a década de 80 como programadores e metodologistas que tentaram diferentes abordagens para análise e projeto orientados a objetos. Entre 1989 e 1994, a quantidade de linguagens de modelagem aumentou de menos de 10 para mais de 50. Esta riqueza de escolha fez com que usuários de métodos orientados a objetos encontrassem dificuldades em determinar uma única linguagem de modelagem que satisfizesse todas as suas necessidades. Em meados da década de 90, novas linguagens de modelagens começaram a assemelhar-se com as anteriores, incorporando elementos uns dos outros. Alguns métodos emergiram, como o Método de Booch, OMT (*Object Modeling Technique*) e OOSE (*Object-Oriented Software Engineering*), cada um desses tem as suas próprias características. O OOSE é orientado a casos de uso e é muito útil à engenharia e à análise de negócio. O OMT expressa bem a análise e os produtos de software de informação. O método de Booch é interessante durante a Fase de Projeto.

Em outubro de 94, Booch e Rumbaugh da Rational começaram a trabalhar na unificação de seus métodos, resultando em um esboço da versão do Método Unificado (*Unified Method*) em outubro de 1995.

Ao mesmo tempo, esforços eram feitos com o objetivo de definir uma linguagem de modelagem padrão. No início de 1995, Jacobson e Richard Soley decidiram investir mais pesadamente para conseguir a padronização no mercado dos métodos. Em junho de 1995, uma reunião do OMG (*Object Management Group*) com todos os principais metodologistas resultou no primeiro acordo para procurar padrões de metodologia.

OOSE foi incorporado ao Método Unificado quando a Objectory foi comprada pela Rational. Seu proprietário, Ivar Jacobson, juntou-se ao trabalho de unificação com Booch e Rumbaugh. As principais razões que motivaram a criação de uma linguagem de modelagem unificada foram:

- Os três métodos estavam evoluindo uma para cada lado independentemente, portanto fazia sentido começar a eliminar as diferenças existentes que não eram significativas, mas potencialmente confusas;
- Juntar os três métodos em um método, estabilizaria o mercado orientado objeto e permitiria que os engenheiros de software se concentrassem nas características mais úteis desta linguagem ao invés de projetar uma nova;
- Eles esperavam que a sua colaboração resultasse em melhorias de aprendizagem e de identificação de soluções aos problemas que não tinham sido resolvidos satisfatoriamente antes.

Booch, Rumbaugh e Jacobson definiram quatro objetivos: i) permitir a modelagem de produtos de software que usam conceitos de orientação a objetos; ii) estabelecer um acoplamento explícito aos artefatos conceituais e executáveis; iii) associar questões de escalabilidade inerentes em produtos de software complexos; e iv) criar uma linguagem de modelagem usável pelos seres humanos e pelas máquinas.

Problemas que tiveram de ser resolvidos dizem respeito ao grau de complexidade envolvendo um *trade-off* entre a expressividade de sua capacidade e a simplicidade que a fazem acessível e fácil de usar. Em junho de 1996, Booch, Rumbaugh e Jacobson liberaram a UML 0.9.

Muitas organizações reconheceram a UML como estratégica para o seu negócio. Elas juntaram forças para responder a um *Request For Proposal* (RFP) emitido pelo OMG. A Rational estabeleceu o consórcio de sócios de UML com as organizações que queriam dedicar recursos para uma definição forte de UML, que incluíam as principais companhias de computador e produtos de software, tais como HP, Microsoft, Oracle, Unisys e IBM. Esta colaboração produziu a UML 1.0, uma linguagem de modelagem bem definida, expressiva, poderosa e aplicável. Foi submetida ao OMG em janeiro de 1997 como uma resposta inicial ao RFP.

Em janeiro de 1997, as respostas ao RFP da IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies e Softeam foram incorporadas pelo consórcio, resultando na UML 1.1, que melhorou a semântica de UML 1.0. As contribuições dos sócios da UML cobriram uma variedade de aspectos, tais como modelagem do negócio, linguagem de restrição, semântica da máquina do estado, tipos, interfaces, componentes, refinamento das colaborações, *frameworks*, distribuição e meta-modelo. Foi submetida ao OMG para sua consideração e adotada em 14 de novembro de 1997.

A manutenção da UML foi então assumida pela RTF (*Revision Task Force*) do OMG, sob a responsabilidade de Cris Kobryn. A RTF lançou uma revisão editorial, a UML 1.2, em junho de 1998. No final do mesmo ano, a RTF lançou a UML 1.3. No ano de 2000, foi lançada a UML 1.4 e, em 2003, surgiu a UML 1.5.

Atualmente, a UML se encontra estável e na versão 2.0, adotada em Outubro de 2004.

A Figura 4.1, ilustrada a seguir, apresenta a evolução da UML.

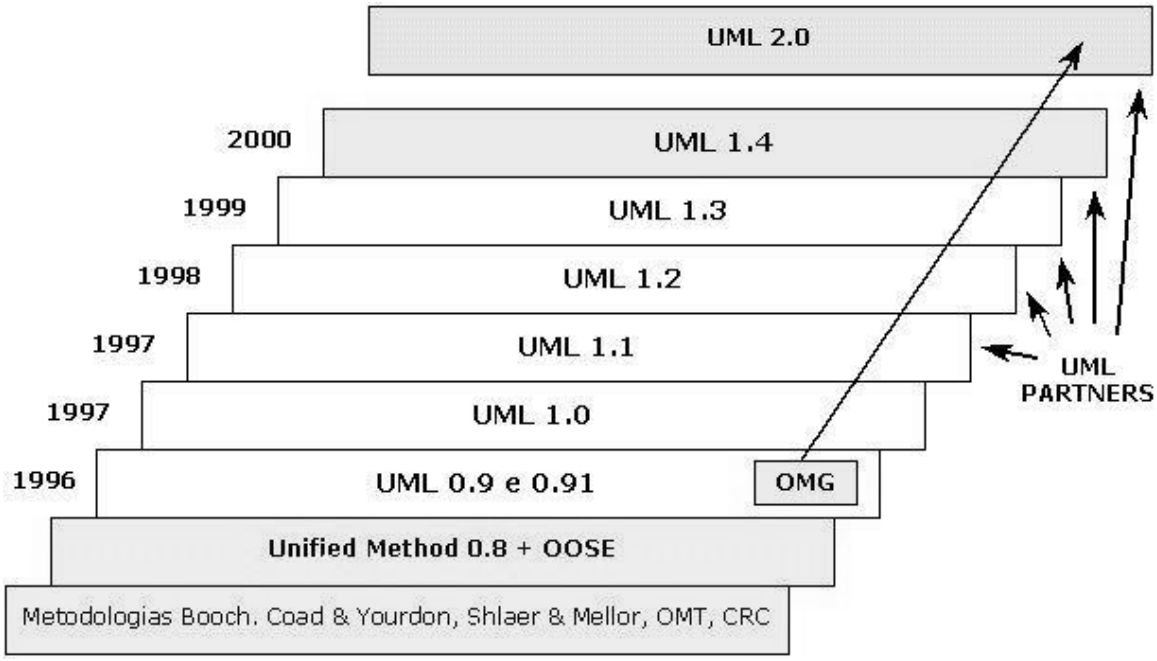


Figura 4.1 – Evolução da UML (Fonte: GEYER, 2005)

4.4 Visão Geral da UML

A UML é uma linguagem de modelagem gráfica destinada a:

- Visualizar, pois facilita a comunicação, algumas estruturas transcendem o que pode ser representado na linguagem de programação e cada elemento de modelagem tem semântica bem definida;
- Especificar, pois detalha decisões importantes na análise, no projeto e na implementação;
- Construir, pois permite engenharia direta (*forward engineering*), engenharia reversa (*reverse engineering*) e engenharia híbrida;
- Documentar, pois artefatos de software gerados incluem documentos de requisitos, especificação funcional e planos de teste, bem como subsídios para controle, medição e comunicação durante o desenvolvimento e após a entrega do produto de software.

A UML se destina principalmente a produtos de software, mas não está restrita a este tipo de modelagem. Na verdade, a UML é suficientemente expressiva para modelar sistemas que não sejam de produtos de software, como um fluxo de trabalho no sistema legal, uma estrutura e um comportamento de sistemas de saúde ou um projeto do hardware.

4.5 Diagramas Propostos pela UML

A notação UML propõe a construção de nove diagramas que são apresentados em BOOCH *et al* (1999). A sua construção se faz necessária à medida que existem pessoas

que possuem capacidade de abstração diferente uma das outras compondo a equipe de desenvolvimento. Portanto, na construção de um produto de software, viabilizar a sua apresentação utilizando diversos ângulos possibilita o seu entendimento pelas pessoas envolvidas qual seja a capacidade de abstração que possuem. BOOCH *et al* (1999) apresenta o seguinte exemplo: dado um conjunto de classes que capturam o propósito do problema, um programador pode precisar de uma visão mais detalhada, no nível de atributos, de operações e de relacionamentos das classes, enquanto um analista que utiliza cenários de casos de uso juntamente com o usuário final precisará de uma visão mais concisa destas classes.

Logo, para usar o modelo a fim de construir uma implementação, ele precisa ser constituído de muitos detalhes. Por outro lado, usar o modelo a fim de apresentar o modelo conceitual para um usuário final, necessita de diagramas que sejam de alto nível de abstração, ou seja, deve esconder os detalhes irrelevantes para a situação.

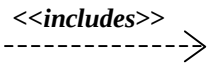
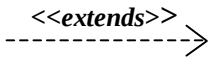
Segundo BOOCH *et al* (1999), os diagramas de UML são divididos em dois grupos: diagramas relacionados à parte estática da modelagem, também conhecidos como diagramas estruturais, e diagramas relacionados à parte dinâmica da modelagem, conhecidos também como diagramas comportamentais. O Diagrama de Classes, o Diagrama de Componentes, o Diagrama de Objetos, o Diagrama de Implantação compõem a parte estática da modelagem. O Diagrama de Atividades, o Diagrama de Casos de Uso, o Diagrama de Máquina de Estados, o Diagrama de Seqüência e o Diagrama de Colaboração compõem a parte dinâmica da modelagem.

4.5.1 Diagrama de Casos de Uso

O Diagrama de Casos de Uso exibe um conjunto de casos de uso, atores e seus relacionamentos. Esse diagrama é usado para descrever e definir os requisitos funcionais de um produto de software. Os atores representam o papel de uma entidade externa ao produto de software como um usuário, um hardware ou outro produto de software que interage com o produto de software modelado. Os atores iniciam a interação através dos casos de uso que representa uma seqüência de ações.

A Tabela 4.1 apresenta elementos comumente utilizados na construção de um Diagrama de Casos de Uso.

Tabela 4.1 – Elementos utilizados em um Diagrama de Casos de Uso

Símbolo	Elemento de Modelagem	Semântica
 <Identificador>	**Ator**	Conjunto de papéis que os usuários dos casos de uso desempenham quando interagem com tais casos de uso, podendo ser uma pessoa, um dispositivo de hardware ou outro sistema automatizado
 <Identificador>	**Caso de Uso**	Representa um conjunto de seqüência de ações que o produto de software desempenha obtendo um resultado observável de valor para um ator
	**Generalização**	Relacionamento entre atores ou entre casos de uso, capaz de especializar tipos genéricos definidos, sendo que o elemento especializado pode adicionar ou modificar o comportamento do elemento genérico
 <<*includes*>>	**Inclusão**	Relacionamento entre casos de uso que indica que um caso de uso deve, obrigatoriamente, incorporar o comportamento definido no outro caso de uso
 <<*extends*>>	**Extensão**	Relacionamento entre casos de uso que indica que um caso de uso pode ou não incorporar o comportamento definido no outro caso de uso
	Associação	Único relacionamento possível entre atores e casos de uso, indicando a comunicação entre eles

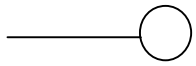
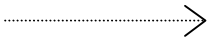
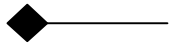
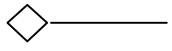
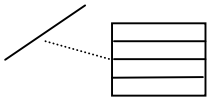
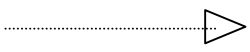
4.5.2 Diagrama de Classes

O Diagrama de Classes é encontrado com maior freqüência na modelagem de produtos de software orientados a objetos. Um Diagrama de Classes mostra um conjunto de classes, interfaces, colaborações e seus relacionamentos.

A modelagem da visão estática de um produto de software pode ser feita utilizando um Diagrama de Classes. Essa perspectiva oferece principalmente suporte para os seus requisitos funcionais – os serviços que o produto de software deverá fornecer aos usuários finais.

A Tabela 4.2 apresenta elementos comumente utilizados na construção de um Diagrama Classes.

Tabela 4.2 – Elementos utilizados em um Diagrama de Classes

Símbolo	Elemento de Modelagem	Semântica
<Identificador> <Atributos> <Métodos> <Responsabilidade>	**Classe**	É o conjunto de objetos que compartilham os mesmos atributos, métodos, responsabilidades e relacionamentos.
<Identificador> – Atributo1 – Atributo2 + Método1 # Método2	**+ public – private # protected**	Identifica quem pode acessar a informação presente na classe. *Private* oculta as informações de todos os elementos externos à classe. *Public* permite que todos os elementos externos acessem as informações. *Protected* permite que apenas os elementos que são subclasses acessem as informações.
	**Interface**	É uma coleção de operações que são usadas para especificar um serviço.
	**Dependência**	Indica que uma mudança na especificação de um elemento de modelagem pode afetar um outro elemento de modelagem que o usa.
1 → apenas um 0 – 1 → zero ou um * → muitos 0..* → zero ou muitos 1 .. * → um ou muitos	**Cardinalidade**	Especifica a quantidade de objetos podem estar relacionados com o outro objeto.
	**Generalização**	Relacionamento onde as classes que representam especializações, compartilham a estrutura e o comportamento definido para a classe que representa uma generalização
<Identificador>	**Associação**	É um relacionamento estrutural que possui um identificador e especifica os objetos de uma classe estão conectados com objetos de uma outra classe.
	**Composição**	Representa uma forma de agregação com relacionamento de propriedade forte e coincidente com o tempo de vida com as partes.
	**Agregação**	Representa uma forma especial de associação que especifica um relacionamento partes/todo entre o agregado (todo) e as componentes (parte).
	**Classe Associativa**	Indica que a associação possui, além das propriedades de associação, propriedades de classe.
	**Realização**	Especifica o relacionamento entre uma interface e a classe que possui a implementação dos serviços da interface.

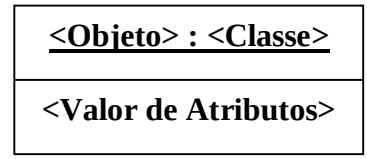
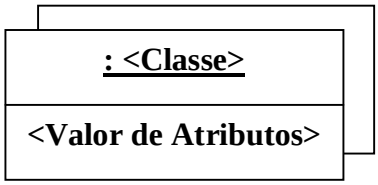
4.5.3 Diagrama de Objetos

O Diagrama de Objetos faz a modelagem de instâncias de itens contidos nos Diagramas de Classes. Um Diagrama de Objetos mostra um conjunto de objetos e seus relacionamentos em determinado ponto do tempo.

A modelagem da visão estática do projeto ou processo de um produto de software pode ser feita utilizando Diagramas de Objetos. Isso envolverá a modelagem de um retrato do produto de software em determinado momento e a representação de um conjunto de objetos, seus estados e relacionamentos.

A Tabela 4.3 apresenta elementos comumente utilizados na construção de um Diagrama Objetos.

Tabela 4.3 – Elementos utilizados em um Diagrama de Objetos

Símbolo	Elemento de Modelagem	Semântica
	**Objeto**	É uma manifestação concreta de uma abstração; uma entidade com limite e identidade bem definidos que encapsula o estado e o comportamento; uma instância de uma classe.
	**Objetos Múltiplos**	Representa objetos múltiplos quando os atributos dos objetos, individualmente, não são importantes.
	Link	Identifica uma colaboração entre os objetos. Conexão semântica entre objetos, representando uma instância de uma associação.

4.5.4 Diagramas de Interação

O Diagrama de Seqüência e o Diagrama de Colaboração – chamados de Diagramas de Interação – são dois dos cinco diagramas utilizados na UML para a modelagem dos aspectos dinâmicos de produtos de software. Um Diagrama de Interação mostra uma interação formada por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que poderão ser enviadas entre eles. Um Diagrama de Seqüência é um Diagrama de Interação que dá ênfase à ordenação temporal das mensagens; o Diagrama de Colaboração é um Diagrama de Interação que dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens.

Como ambos são derivados das mesmas informações de um metamodelo da UML, o Diagrama de Seqüência e o Diagrama de Colaboração são semanticamente equivalentes. Portanto, o diagrama de uma forma pode ser convertido no outro sem qualquer perda de informação.

Diagramas de Interações podem aparecer sozinhos para visualizar, especificar, construir e documentar a dinâmica de uma determinada sociedade de objetos ou podem ser utilizados para fazer a modelagem de determinado fluxo de controle de um caso de uso.

A Tabela 4.4 e a Tabela 4.5 apresentam, respectivamente, elementos comumente utilizados na construção de um Diagrama de Seqüência e de um Diagrama de Colaboração.

Tabela 4.4 – Elementos utilizados em um Diagrama de Seqüência

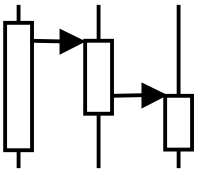
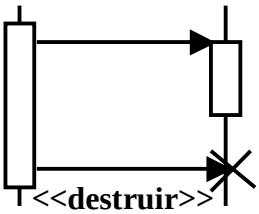
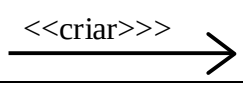
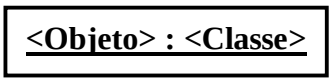
Símbolo	Elemento de Modelagem	Semântica	
<Objeto> : <Classe>	**Objeto**	Representa o objeto que participa da função descrita pelo diagrama. Usa-se o símbolo de objeto sem a sua lista de atributos e métodos.	
	**Foco de Controle**	É representada por uma faixa e corresponde ao período de tempo em que um objeto executa sua tarefa, após ser ativado.	
	Linha de Vida	Representa a existência de um objeto em um período de tempo.	
	**Mensagem**	**Síncrono** **Normal** **Assíncrono**	É representada por uma seta horizontal e corresponde à troca de mensagens entre objetos.
	**Destruição de Objeto**	Objetos podem ser destruídos usando uma seta com o rótulo "<<**destruir**>>" apontando para um X.	
	**Criação do Objeto**	Objetos podem ser criados usando uma seta com o rótulo "<<**criar**>>".	

Tabela 4.5 – Elementos utilizados em um Diagrama de Colaboração

Símbolo	Elemento de Modelagem	Semântica
	**Objeto**	Representa o objeto que participa da função descrita pelo diagrama. Usa-se o símbolo de objeto sem a sua lista de atributos e métodos.
	Link	Identifica uma colaboração entre os objetos. Conexão semântica entre objetos, representando uma instância de uma associação.
	**Mensagem**	É representada por uma seta horizontal e corresponde à troca de mensagens entre objetos.

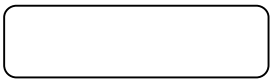
4.5.5 Diagrama de Estados

O Diagrama de Estados é empregado para a modelagem dos aspectos dinâmicos de um produto de software. Na maior parte, isso envolve a modelagem do comportamento de objetos reativos, cujo comportamento é mais bem caracterizado por sua resposta a eventos ativados externamente ao seu contexto. O Diagrama de Estados pode ser anexado a classes, a casos de uso ou a produtos de software inteiros para visualizar, especificar, construir e documentar a dinâmica de um objeto individual.

Especificamente, um Diagrama de Estados mostra uma máquina de estados, dando ênfase ao fluxo de controle de um estado para outro. Uma máquina de estados é um comportamento que especifica as seqüências de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos. Um estado é uma condição ou situação na vida de um objeto durante a qual ele satisfaz a alguma condição, realiza alguma atividade ou aguarda algum evento. Um evento é uma especificação de uma ocorrência significativa que tem uma localização no tempo e no espaço. No contexto de uma máquina de estados, um evento é uma ocorrência de um estímulo capaz de ativar a transição de estado. Uma transição é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará no segundo estado quando um evento especificado ocorrer se as condições especificadas estão satisfeitas.

A Tabela 4.6 apresenta elementos comumente utilizados na construção de um Diagrama Estados.

Tabela 4.6 – Elementos utilizados em um Diagrama de Estados

Símbolo	Elemento de Modelagem	Semântica
	**Estado**	Representa o estado do objeto durante a sua existência.
<evento> **[Condição]** **/<ação>** 	**Transição**	Representa um relacionamento entre dois estados, indicando que o objeto no primeiro estado desempenhará certas ações e entrará no segundo estado quando um evento específico ocorre e a condição estipulada for atendida.
[Condição]	**Condição de Guarda**	Especifica uma expressão *booleana*, cujo valor permite ou não a transição.
	**Início**	Representa o estado inicial.
	**Término**	Representa o estado final.

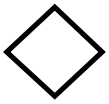
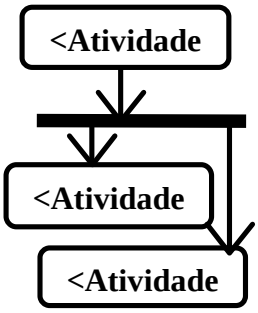
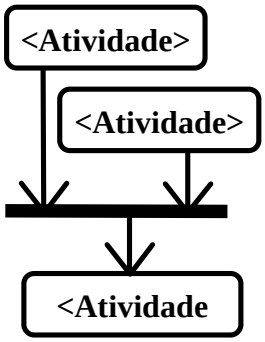
4.5.6 Diagrama de Atividades

Um Diagrama de Atividades é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra. O Diagrama de Atividades é empregado para fazer a modelagem dos aspectos dinâmicos do produto de software. Na maior parte do tempo, isso envolve a modelagem das etapas seqüenciais em um processo computacional. Com um Diagrama de Atividade, é possível fazer a modelagem do fluxo de um objeto, à medida que ele passa de um estado para outro em pontos diferentes do fluxo de controle. Os diagramas poderão permanecer isolados, para visualizar, especificar, construir e documentar a dinâmica de uma sociedade de objetos, ou poderão ser utilizados para fazer a modelagem do fluxo de controle de uma operação.

Diferente dos Diagramas de Interação que enfatizam o fluxo de controle de um objeto para outro, os Diagramas de Atividades dão ênfase ao fluxo de uma atividade para outra.

A Tabela 4.7 apresenta elementos comumente utilizados na construção de um Diagrama Atividades.

Tabela 4.7 – Elementos utilizados em um Diagrama de Atividades

Símbolo	Elemento de Modelagem	Semântica
	**Atividade ou Ação**	Atividade é uma execução não atômica que resulta em alguma ação. Ação, por sua vez, é uma computação atômica que resulta em uma mudança no estado do sistema ou no retorno de um valor.
	**Transição**	Relacionamento que mostra o fluxo de controle sendo passado de uma atividade ou ação para outra.
[Condição]	**Condição de Guarda**	Especifica uma expressão *booleana*, cujo valor permite ou não a transição.
	**Desvio**	Representa desvio para caminhos alternativos que devem ser seguidos em decorrência do resultado de uma expressão *booleana* (condição de guarda).
	**Início**	Indica o local de início *default*.
	**Término**	Indica que a execução do diagrama foi completada.
	**Separação**	Representa a segmentação de um fluxo de controle (atividade) em dois ou mais fluxos de controle, que são atividades que continuam a sua execução em paralelo.
	**Junção**	Representa a junção de dois ou mais fluxos de controle (atividades) em apenas um único fluxo de controle (atividade).

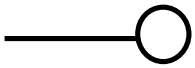
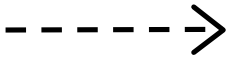
4.5.7 Diagrama de Componentes

Um componente é uma parte física e substituível de um produto de software ao qual se adapta e fornece a realização de um conjunto de interfaces. Uma interface é uma coleção de operações utilizadas para especificar um serviço de uma classe ou de um componente.

O Diagrama de Componentes é empregado para a modelagem da visão estática de implementação de um produto de software. Isso envolve a modelagem de itens físicos que residem em um nó, como executáveis, bibliotecas, tabelas, arquivos e documentos.

A Tabela 4.8 apresenta elementos comumente utilizados na construção de um Diagrama Componentes.

Tabela 4.8 – Elementos utilizados em um Diagrama de Componentes

Símbolo	Elemento de Modelagem	Semântica
	Componente	São unidades físicas de implementação com um conjunto bem definido de interfaces que atuam como trechos substituíveis dentro do produto de software.
	Interface	São coleções de operações usadas para especificar um serviço. A interface que um componente realiza é interface de exportação. A interface que um componente usa é interface de importação.
	Relacionamento de Realização/ Relacionamento de Dependência	Relacionamento capaz de indicar que um componente utiliza informações definidas em outro componente.
	Biblioteca	Tipo de componente para representar uma biblioteca estática ou dinâmica.
	Tabela	Tipo de componente para representar uma tabela.
	Arquivo	Tipo de componente para representar um arquivo.
	Documento	Tipo de componente para representar um documento.
	Banco de Dados	Tipo de componente para representar um banco de dados.

4.5.8 Diagrama de Implantação (*Deployment*)

O Diagrama de Implantação é empregado para a modelagem da visão estática da implantação de um produto de software. Isso envolve a modelagem da topologia do hardware em que o produto de software é executado.

Um Diagrama de Implantação é composto por componentes, que possuem a mesma simbologia e semântica definida para o Diagrama de Componentes, nós, que significam

objetos físicos que fazem parte do produto de software, podendo ser uma máquina cliente numa LAN, uma máquina servidora, uma impressora, um roteador, etc., e conexões entre estes nós e componentes que juntos compõe toda a arquitetura física do produto de software.

A Tabela 4.9 apresenta elementos comumente utilizados na construção de um Diagrama de Implantação (Deployment).

Tabela 4.9 – Elementos utilizados em um Diagrama de Implantação (Deployment)

Símbolo	Elemento de Modelagem	Semântica
	Nó	É um elemento físico que existe em tempo de execução e que representa um recurso computacional, geralmente, tem pelo menos memória e, algumas vezes, capacidade de processamento.
————	Relacionamento de Associação	É o relacionamento entre nós, também chamado de conexão.

4.6 Considerações Finais

O presente capítulo apresentou uma breve história da UML, a importância da modelagem, uma visão geral e uma descrição sucinta dos diagramas propostos pela linguagem.

O desenvolvimento da UML foi baseado em técnicas antigas e marcantes da orientação a objetos e, atualmente, a linguagem de modelagem é uma tendência entre desenvolvedores.

A UML proporciona às empresas de desenvolvimento de produtos de software uma maior comunicação e aproveitamento dos modelos desenvolvidos pelos seus vários analistas, pois a linguagem utilizada por todos é a mesma, acabando assim com qualquer problema de interpretação e mal-entendimento de modelos criados por outros desenvolvedores.

Ao se utilizar a UML, atendendo a metodologia da linguagem de forma correta, empresas e desenvolvedores atingirão seus objetos, desenvolvendo produtos de software de forma rápida, eficiente e efetiva.

5 MODELAGEM DE DADOS UTILIZANDO O MODELO ENTIDADE RELACIONAMENTO

5.1 Considerações Iniciais

Dados são fatos conhecidos que podem ser registrados e que possuem significado implícito. Um banco de dados é uma coleção de dados relacionados, possuindo as seguintes propriedades (ELMASRI & NAVATHE, 2002):

1. Um banco de dados representa algum aspecto do mundo real, algumas vezes chamado de minimundo (*miniworld*) ou de Universo de Discurso (*UoD, Universe of Discourse*). Alterações no minimundo são refletidas no banco de dados;
2. Um banco de dados é uma coleção lógica e coerente de dados com algum significado inerente. Uma organização aleatória de dados não pode ser referenciada corretamente como um banco de dados;
3. Um banco de dados é projetado, construído e povoado com dados que possuem um objetivo específico. Ele possui um grupo provável de usuários e algumas aplicações preconcebidas, nas quais esses usuários estão interessados.

Em outras palavras, um banco de dados possui alguma fonte, da qual os dados são derivados, algum grau de interação com eventos do mundo real e um público que está ativamente interessado no conteúdo do banco de dados.

5.2 Modelagem de Dados Conceitual

Uma característica fundamental do enfoque de banco de dados é o fato de que ele oferece algum nível de abstração de dados, escondendo detalhes de armazenamento de dados que não são necessários para a maioria dos usuários. Um modelo de dados – uma coletânea de conceitos que podem ser utilizados para descrever a estrutura de um banco de dados – fornece os meios necessários para alcançar essa abstração. ELMASRI & NAVATHE (2002) define estrutura de um banco de dados como os tipos de dados, relacionamentos e restrições que devem existir nos dados. A maioria dos modelos de dados também inclui um conjunto de operações básicas para especificar recuperações e atualizações no banco de dados.

Muitos modelos de dados foram propostos e, segundo ELMASRI & NAVATHE (2002), podem ser categorizados de acordo com os tipos de conceitos que eles utilizam

para descrever a estrutura de um banco de dados. Modelos Conceituais oferecem conceitos que estão próximos do modo como muitos usuários percebem os dados, enquanto que os Modelos Físicos de dados oferecem conceitos que descrevem os detalhes de como os dados estão armazenados no computador.

Os Modelos Conceituais utilizam conceitos como entidades, atributos e relacionamentos. Uma entidade representa um objeto ou conceito do mundo real, como um empregado ou um projeto que está descrito no banco de dados. Um atributo representa alguma propriedade ou característica de interesse que descreve uma entidade, como o nome ou o salário do empregado. Um relacionamento entre duas ou mais entidades representa uma interação entre as entidades; por exemplo, um relacionamento "trabalha em" entre um empregado e um projeto.

5.3 Modelo Entidade-Relacionamento

O Modelo Entidade-Relacionamento (MER) é um modelo de dados conceitual. Assim, os conceitos do MER foram projetados para serem compreensíveis aos usuários, descartando detalhes de como os dados são armazenados.

Atualmente, o MER é utilizado principalmente durante o processo de projeto de banco de dados.

O MER tem por base a percepção de que o mundo real é formado por entidades e relacionamentos entre elas. O MER é um dos modelos com maior capacidade semântica, ou seja, os aspectos semânticos do modelo se referem à tentativa de representar o significado dos dados. O MER é representado graficamente através do Diagrama Entidade-Relacionamento (DER).

5.3.1 Entidades

O objeto básico que o MER representa é uma entidade, que representa algo do mundo real com uma existência independente. Uma entidade pode ser um objeto com uma existência física – uma determinada pessoa, carro, casa ou empregado – ou pode ser um objeto com uma existência conceitual – uma empresa, um serviço ou um curso numa universidade (ELMASRI & NAVATHE, 2002).

5.3.2 Atributos

Cada entidade possui atributos – as propriedades específicas que as descrevem. Por exemplo, uma entidade empregado pode ser descrita através do nome, idade, endereço, salário e profissão. Os valores dos atributos que descrevem cada entidade tornam-se uma parte importante dos dados armazenados no banco de dados. Os atributos podem ser caracterizados pelos seguintes tipos (ELMASRI & NAVATHE, 2002):

- Compostos x Simples. Os atributos compostos podem ser divididos em outros atributos básicos com significados diferentes. Por exemplo, o atributo Endereço da entidade empregado pode ser subdividido em NomeDoLogradouro, NúmeroDoLogradouro, Cidade, Estado e Caixa Postal, com os valores "Avenida Central", "95", "Lavras", "MG" e "37200-000". Os atributos simples são atômicos, ou seja, não são divisíveis. Por exemplo, o atributo Cidade;
- Monovalorados x Multivalorados. A maioria dos atributos possui um valor único para uma determinada entidade; esses atributos são chamados de monovalorados. Por exemplo, o atributo Idade é um atributo de valor único da pessoa. Em alguns casos, um atributo pode ter um conjunto de valores para a mesma entidade – por exemplo, um atributo Cores para um carro. Carros com uma cor possuem valor único; enquanto carros com duas tonalidades possuem dois valores para Cores. Esses atributos são chamados de multivalorados;
- Primitivo x Derivado. Em alguns casos, dois (ou mais) valores de atributos estão relacionados – por exemplo, os atributos Idade e DataDeNascimento de uma pessoa. Para uma determinada entidade pessoa, o valor de Idade pode ser determinado a partir da data atual (hoje) e o valor da DataDeNascimento daquela pessoa. O atributo Idade é portanto chamado de atributo derivado e diz-se que é derivável do atributo DataDeNascimento, que é chamado de atributo primitivo;
- Nulo. Uma determinada entidade pode não ter um valor aplicável para um atributo. Por exemplo, o atributo NúmeroDoApartamento de um endereço se aplica somente a endereços que se encontram em prédios e apartamentos.

Uma entidade tem atributos cujo valor é distinto para cada entidade. Um atributo deste tipo é chamado de chave e seu valor pode ser usado para identificar cada entidade unicamente. Às vezes, um conjunto de atributos pode formar uma chave.

A especificação de um atributo chave para uma entidade significa que a propriedade de unicidade deve valer para quaisquer extensões desta entidade. Assim, esta restrição proibe que duas entidades tenham o mesmo valor para o atributo chave.

5.3.3 Relacionamentos

Relacionamento é uma associação de uma ou mais entidades (ELMASRI & NAVATHE, 2002).

Nos DER, os relacionamentos são exibidos como caixas em formato de diamante (losangos), que são ligadas através de linhas retas às caixas retangulares que representam as entidades participantes do relacionamento. O nome do relacionamento é exibido na caixa com formato de diamante.

Os relacionamentos geralmente possuem certas restrições que limitam as possíveis combinações de entidades que podem participar no correspondente conjunto de relacionamentos. Segundo ELMASRI & NAVATHE (2002), pode-se distinguir dois tipos principais de restrições em relacionamentos:

- Razões de cardinalidade para Relacionamentos Binários. Especifica o número de instâncias do relacionamento nas quais uma entidade pode participar. Por exemplo, o relacionamento binário EMPREGADO TRABALHA_EM DEPARTAMENTO possui a razão de cardinalidade 1:N, significando que cada empregado pode ser relacionado a apenas um departamento e um departamento pode possuir inúmeros empregados.
- Restrições de participação e Dependências de Existências. Especifica se a existência de uma entidade depende dela ser relacionada a uma outra entidade através do tipo de relacionamento. Existem dois tipos de restrições de participação – total e parcial. A restrição de participação total estabelece que uma instância da entidade E_1 somente pode existir se possuir vínculo com alguma instância da entidade E_2 no mesmo relacionamento. A participação total também é chamada de dependência de existência. A restrição de participação parcial estabelece que algumas instâncias da entidade E_1 possuem vínculo com alguma instância da entidade E_2 no mesmo relacionamento.

5.3.3.1 Notação

A Figura 5.1 ilustra um resumo das notações para a construção do DER.

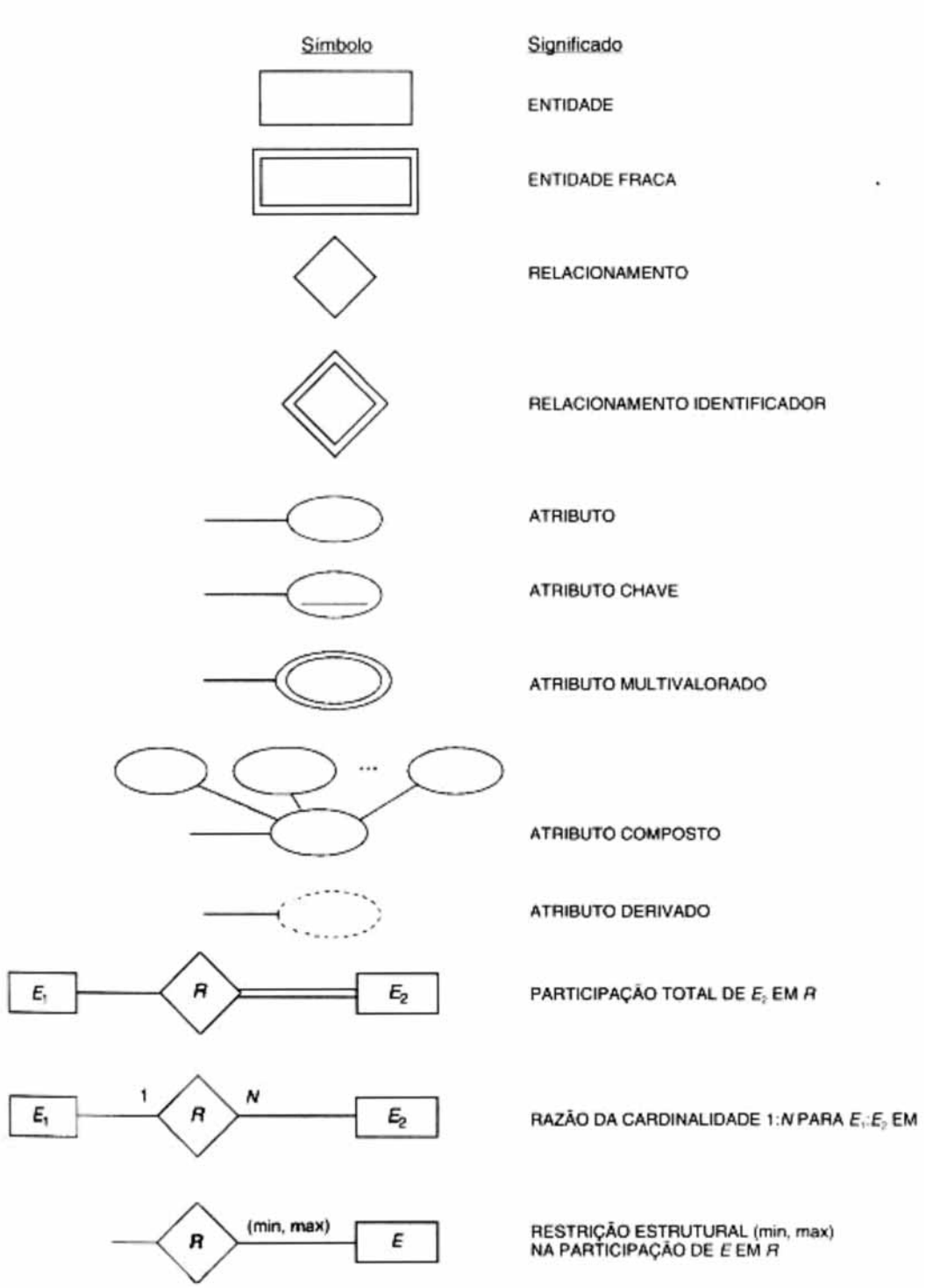


Figura 5.1 – Elementos de Modelagem para Construir o DER (Fonte: ELMASRI & NAVATHE, 2002)

5.4 Considerações Finais

Neste capítulo, foram apresentados os conceitos de Modelagem Entidade-Relacionamento, descrevendo os conceitos básicos de restrições e estruturação de dados do MER. Além disso, foram apresentados os elementos de modelagem para construir o DER.

Por descrever a estrutura de um banco de dados de uma forma compreensível a usuários, a modelagem de dados conceitual se torna indispensável no desenvolvimento de produtos de software.

Uma das formas de modelar um banco de dados é utilizando o MER, um modelo de dados conceitual de alto nível bastante popular que possui variações freqüentemente utilizadas para o projeto conceitual de aplicações de bancos de dados (ELMASRI & NAVATHE, 2002).

6 OS CASOS CPPD E CIS

6.1 Considerações Iniciais

Os produtos de software CPPD (Comissão de Pessoal Permanente Docente) e CIS (Comissão Interna de Supervisão) desenvolvidos neste trabalho garantem alta portabilidade, podendo ser utilizados em vários sistemas operacionais, bem como diferentes plataformas de hardware.

Nas seções a seguir, são descritos como os produtos de software foram desenvolvidos.

6.2 Metodologia de Pesquisa

Para a manutenção do CPPD e desenvolvimento do CIS foram consideradas as seguintes fases de desenvolvimento de produtos de software: Análise de Requisitos, Análise, Projeto, Programação e Testes, considerando o Modelo Interativo e Incremental.

Segundo EINSENMANN & MARTINS (2005):

- Análise de Requisitos captura as intenções e as necessidades dos usuários do produto de software a ser desenvolvido através do uso de funções chamadas Casos de Uso. Através do desenvolvimento de cada caso de uso, os atores externos ao produto de software que interagem e possuem interesse são modelados entre as funções que eles requerem. Nesta fase inicial, realizaram-se reuniões com o cliente de modo a coletar o máximo de informações possíveis acerca do funcionamento dos produtos de software e, ao seu término, foram gerados o Diagrama de Casos de Uso e a Descrição dos Casos de Uso;
- Análise está preocupada com as primeiras abstrações (classes e objetos) e mecanismos que estarão presentes no domínio do problema. As classes são modeladas e ligadas através de relacionamentos com outras classes gerando o Diagrama de Classes e são descritas no Dicionário de Atributos e Métodos. As colaborações entre classes também são mostradas neste diagrama para desenvolver os casos de uso modelados anteriormente. Na análise, serão modeladas classes que pertençam a lógica de negócio a ser automatizado;
- Projeto expande o resultado da análise em soluções técnicas. Novas classes serão adicionadas para prover uma infra-estrutura que tenha característica de implementação, por exemplo, interface do usuário e de periféricos, gerenciamento de banco de dados e

comunicação com outros sistemas. As classes do domínio do problema modeladas na análise são mescladas nessa nova infra-estrutura. O projeto resulta no detalhamento das especificações para a programação do produto de software. Optou-se por desenvolver os produtos de software utilizando três camadas, que são Interface, Negócio e Persistência, visando facilitar a posterior manutenção. A camada de Interface é responsável por prover comunicação com o usuário do sistema, obtendo os dados fornecidos, validando-os e fornecendo-os para a camada de Negócio. A camada de Negócio, como o próprio nome indica, modela o domínio do negócio. A camada de Persistência tem o objetivo de recuperar e armazenar objetos permanentemente em meio físico;

- Na Programação, as classes provenientes do projeto são convertidas para uma linguagem de programação. No contexto deste trabalho, a linguagem utilizada foi Java;
- Por fim, são realizados diversos testes com o intuito de verificar se as classes estão cooperando uma com as outras como especificado nos modelos e se o produto de software está funcionando como o especificado no Diagrama de Casos de Uso, construído na Análise de Requisitos.

6.3 Produto de Software: CPPD

A Comissão Permanente de Pessoal Docente da Universidade Federal de Lavras possui, dentre suas funções, o objetivo de manter um cadastro dos docentes da universidade, bem como controlar o regime de trabalho e avaliar o desempenho de cada docente para sua progressão funcional.

O produto de software CPPD visa facilitar este cadastro e o controle dos demais processos dos docentes, sendo possível também obter informações relevantes através de relatórios mais elaborados, o que não ocorria anteriormente.

6.3.1 Migração de Plataforma

O produto de software CPPD havia sido previamente desenvolvido em linguagem de programação Object Pascal, utilizando-se Delphi como ambiente de desenvolvimento. Seus dados estavam armazenados no Paradox, um SGBD incluso no Delphi. Nenhuma documentação acerca deste produto de software havia sido criada. Além disso, o produto de software e o banco de dados estavam na mesma máquina, impossibilitando acesso aos dados externamente à rede da Diretoria de Recursos Humanos (DRH).

Desta forma, houve a necessidade de gerar toda a documentação do sistema, migrá-lo para linguagem de programação Java e migrar os dados para o SGBD MySQL, possibilitando qualquer outro produto de software em qualquer máquina com acesso à Internet, mediante autenticação, recuperar dados, que se encontram em um computador utilizado exclusivamente como servidor de banco de dados.

Baseado na técnica de Engenharia Reversa foi realizado um estudo minucioso do código-fonte do antigo produto de software CPPD e reuniões com seu desenvolvedor. Com o conhecimento sedimentado, foi criada uma representação do sistema em um nível de abstração superior e iniciou-se a manutenção que, nesse contexto, poderia ser caracterizada como adaptativa, preventiva e corretiva. A manutenção consistiu em reestruturar o produto de software, desenvolvendo um novo produto com base na representação gerada, utilizando tecnologia mais avançada.

A migração dos dados foi realizada desenvolvendo um aplicativo em Object Pascal, que transferiu todo o conteúdo do Paradox para arquivos de texto com formato previamente definido. Em seguida, foi desenvolvido um aplicativo em Java, que transferiu todo o conteúdo dos arquivos de texto para o banco de dados no MySQL, finalizando, assim, a migração de plataforma.

6.3.2 Modelagem dos Dados

A Figura 6.1 apresenta o Modelo Entidade Relacionamento do produto de software CPPD.

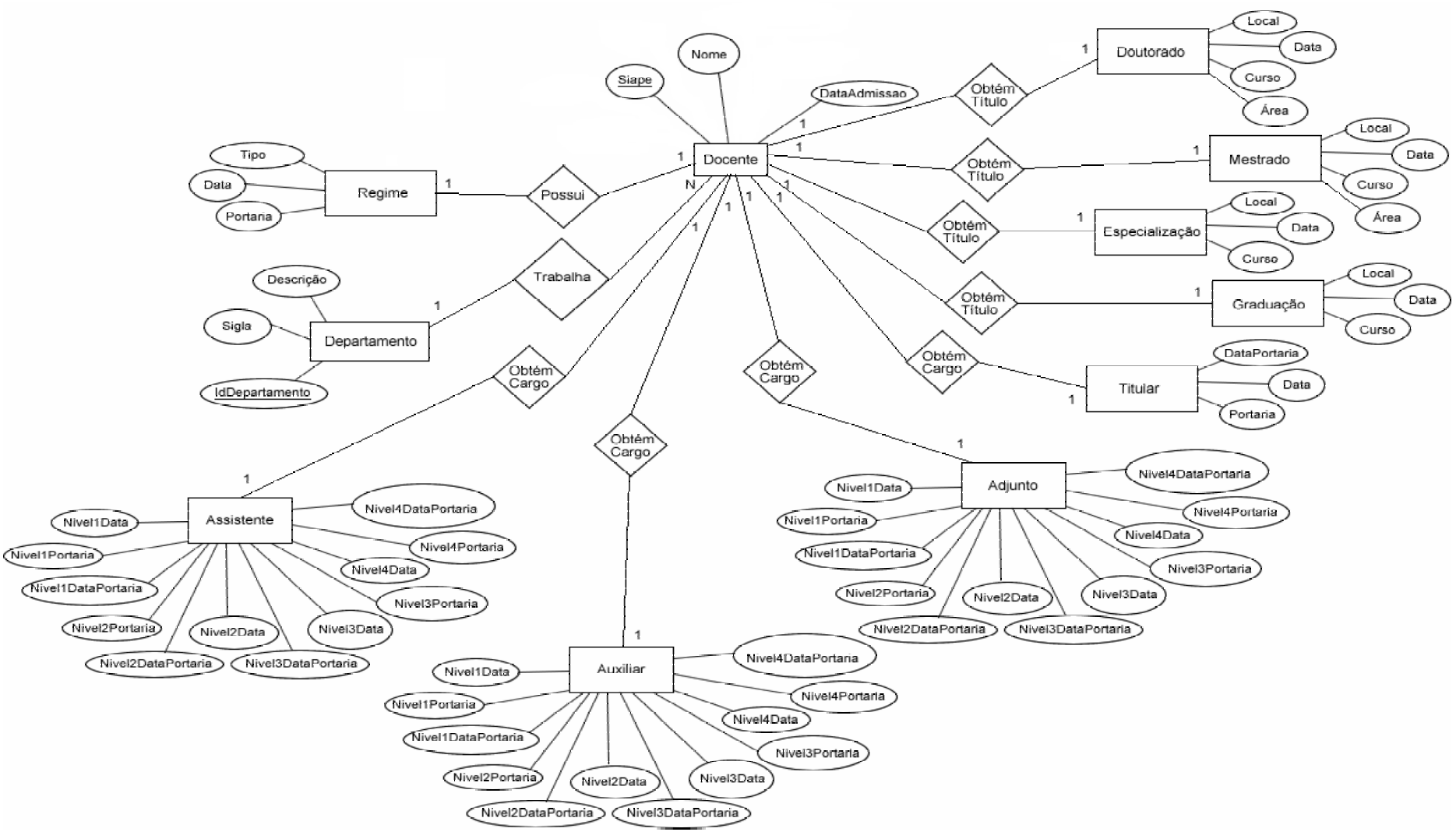


Figura 6.1 – Modelo Entidade-Relacionamento – CPPD

6.3.3 Desenvolvimento

Nesta seção, é descrito como o produto de software CPPD foi desenvolvido, descrevendo sucintamente as fases do processo de desenvolvimento pelo qual ele passou.

6.3.3.1 Análise de Requisitos

Durante esta fase, foram construídos o Diagrama de Casos de Uso e a Descrição dos Casos de Uso descritos nas seções subseqüentes.

6.3.3.1.1 Diagrama de Casos de Uso

A Figura 6.2 ilustra o Diagrama de Casos de Uso com o ator funcionário e os casos de uso do Produto de Software CPPD: Manter Cadastro de Departamento, Manter Cadastro de Docente, Modificar Progressão de Docente, Consultar Progressão de Docente, Modificar Regime de Docente, Consultar Regime de Docente, Emitir Relatório por Cargo, Emitir Relatório por Departamento, Emitir Relatório por Titulação, Emitir Relatório por Data de Admissão e Emitir Relatório por Progressão do Mês.

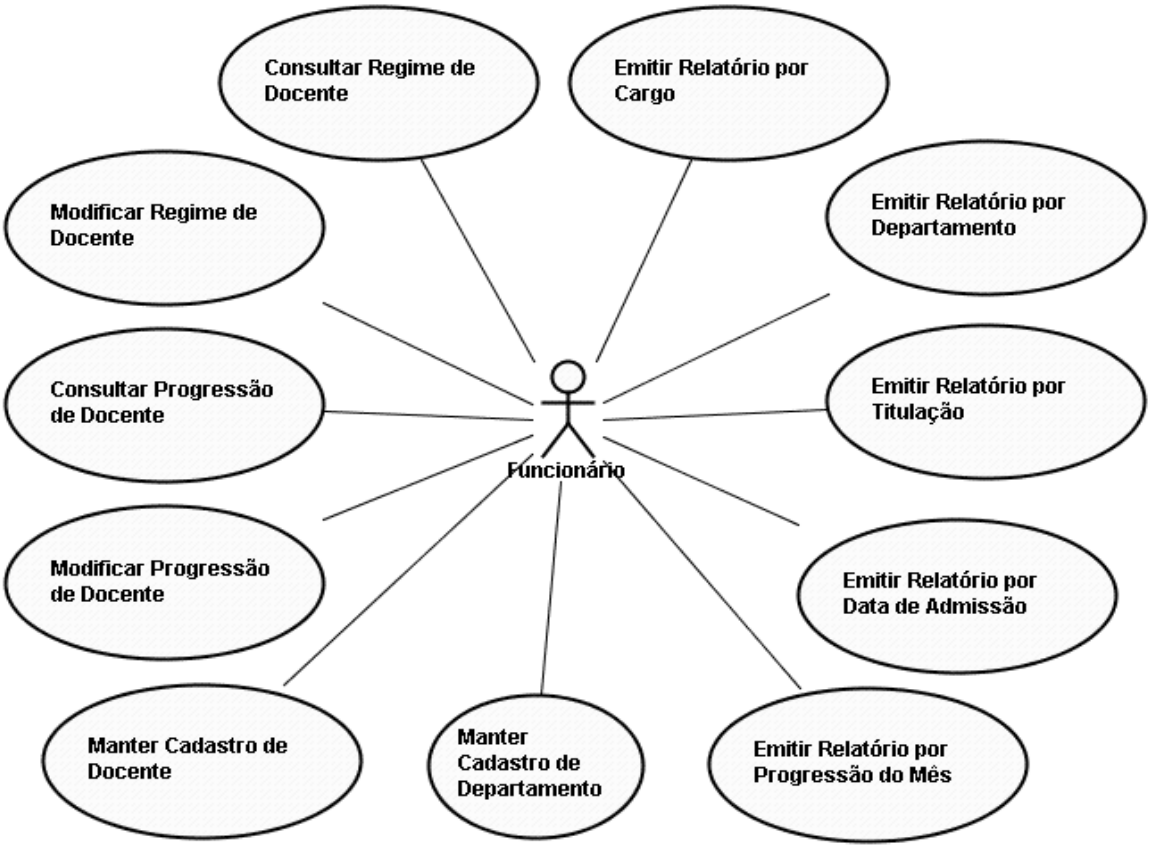


Figura 6.2 – Diagrama de Casos de Uso

O caso de uso Manter Cadastro de Departamento engloba as operações Inserir, Alterar e Excluir. O caso de uso Manter Cadastro de Docente contém todas as anteriores, acrescidas da operação Consultar.

Para a representação do Diagrama de Casos de Uso, foram utilizados os elementos de modelagem sugeridos pela UML (BOOCH *et al*, 1999).

6.3.3.1.2 Descrição dos Casos de Uso

A Descrição dos Casos de Uso tem por objetivo registrar de forma textual, o comportamento de cada caso de uso, acrescentando detalhes que enriquecem e tornam mais claros e completos os casos de uso. Estas informações são:

- **Id**: Identificador numérico único;
- **Nome**: Nome único para descrever a função do caso de uso;
- **Descrição**: Descrição sucinta das atividades que devem ser consideradas;
- **Ator**: Nome do ator(es) que interage(m) com o produto de software;
- **Pré-Condições**: Condições em que o produto de software de estar para iniciar o caso de uso principal;
- **Execução Normal**: Seqüência de passos que devem ocorrer quando o produto de software atender a todas as pré-condições, com as mensagens necessárias ao(s) ator(es);
- **Pós-Condições**: Condições em que o produto de software deve estar após a execução do caso de uso;
- **Exceções**: seqüência de passos que devem ocorrer quando alguma pré-condição não for verificada, com as mensagens necessárias ao(s) ator(es).

A seguir, é apresentada a Descrição do Caso de Uso dos seguintes casos de uso:

- Manter Cadastro de Departamento (Tabela 6.1);
- Manter Cadastro de Docente (Tabela 6.2);
- Modificar Progressão de Docente (Tabela 6.3);
- Consultar Progressão de Docente (Tabela 6.4);
- Modificar Regime de Docente (Tabela 6.5);
- Consultar Regime de Docente (Tabela 6.6);
- Emitir Relatório por Cargo (Tabela 6.7);
- Emitir Relatório por Departamento (Tabela 6.8);
- Emitir Relatório por Titulação (Tabela 6.9);

- Emitir Relatório por Data de Admissão (Tabela 6.10);
- Emitir Relatório por Progressão do Mês (Tabela 6.11).

Tabela 6.1 – Descrição do Caso de Uso Manter Cadastro de Departamento

Id Caso de Uso: 001	
Nome: Manter Cadastro de Departamento	
Descrição: Permite realizar as operações de Inclusão, Alteração e Exclusão de Departamentos.	
Ator: Funcionário	
Incluir Departamento	
Pré-Condições	O departamento não deve existir no sistema.
Execução Normal	A inclusão do departamento é feita com a entrada de seus dados.
Pós-Condições	O sistema emite uma mensagem indicando que a inclusão foi efetuada com sucesso.
Exceções	Se o departamento existir no sistema, a inclusão não é realizada e o sistema emite uma mensagem indicando que a inclusão não foi efetuada.
Alterar Departamento	
Pré-Condições	O funcionário deve selecionar o departamento a ser alterado.
Execução Normal	O funcionário altera os dados do departamento previamente selecionado.
Pós-Condições	O sistema emite uma mensagem indicando que as alterações foram efetuadas com sucesso.
Exceções	Se o funcionário não selecionar algum departamento, é emitida uma mensagem indicando que algum departamento deve ser selecionado.
Excluir Departamento	
Pré-Condições	O funcionário deve selecionar o departamento a ser excluído.
Execução Normal	O funcionário seleciona o departamento a ser excluído.
Pós-Condições	O sistema emite uma mensagem indicando que o departamento foi excluído com sucesso.
Exceções	Se o funcionário não selecionar algum departamento, é emitida uma mensagem indicando que algum departamento deve ser selecionado.

Tabela 6.2 – Descrição do Caso de Uso Manter Cadastro de Docente

Id Caso de Uso: 002	
Nome: Manter Cadastro de Docente	
Descrição: Permite realizar as operações de Inclusão, Alteração, Exclusão e Consulta de Docentes.	
Ator: Funcionário	
Incluir Docente	
Pré-Condições	O docente não deve existir no sistema.
Execução Normal	A inclusão do docente é feita com a entrada de seus dados.
Pós-Condições	O sistema emite uma mensagem indicando que a inclusão foi efetuada com sucesso. Automaticamente, são criados sua

	progressão e seu regime.
Exceções	Se o docente existir no sistema, a inclusão não é realizada e o sistema emite uma mensagem indicando que a inclusão não foi efetuada.
Alterar Docente	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do docente, alterando os dados desejados posteriormente.
Pós-Condições	O sistema emite uma mensagem indicando que as alterações foram efetuadas com sucesso.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.
Excluir Docente	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do docente. Uma mensagem de confirmação é exibida. Automaticamente, são excluídos sua progressão e seu regime.
Pós-Condições	O sistema emite uma mensagem indicando que a exclusão foi concluída com sucesso.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.
Consultar Docente	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	O funcionário realiza uma consulta sobre algum docente informando seu número siape ou seu nome.
Pós-Condições	Os dados do docente são exibidos.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.

Tabela 6.3 – Descrição do Caso de Uso Modificar Progressão de Docente

Id Caso de Uso: 003	
Nome: Modificar Progressão de Docente	
Descrição: Permite modificar a progressão dos docentes cadastrados.	
Ator: Funcionário	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	Após informar o número siape ou nome do docente, é possível modificar os dados de sua progressão.
Pós-Condições	É emitida uma mensagem indicando que seus dados foram modificados com sucesso.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.

Tabela 6.4 – Descrição do Caso de Uso Consultar Progressão de Docente

Id Caso de Uso: 004
Nome: Consultar Progressão de Docente
Descrição: Permite consultar a progressão dos docentes cadastrados.

Ator: Funcionário	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do docente.
Pós-Condições	Os dados da progressão do docente são exibidos.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.

Tabela 6.5 – Descrição do Caso de Uso Modificar Regime de Docente

Id Caso de Uso: 005	
Nome: Modificar Regime de Docente	
Descrição: Permite modificar o regime dos docentes cadastrados.	
Ator: Funcionário	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	Após informar o número siape ou o nome do docente, é possível modificar os dados de seu regime.
Pós-Condições	É emitida uma mensagem indicando que seus dados foram modificados com sucesso.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.

Tabela 6.6 – Descrição do Caso de Uso Consultar Regime de Docente

Id Caso de Uso: 006	
Nome: Consultar Regime de Docente	
Descrição: Permite consultar o regime dos docentes cadastrados.	
Ator: Funcionário	
Pré-Condições	O docente deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do docente.
Pós-Condições	Os dados do regime do docente são exibidos.
Exceções	Se o docente não existir no sistema, é emitida uma mensagem indicando que o docente não existe.

Tabela 6.7 – Descrição do Caso de Uso Emitir Relatório por Cargo

Id Caso de Uso: 007	
Nome: Emitir Relatório por Cargo	
Descrição: Emite um relatório contendo informações dos docentes que possuem o mesmo cargo. Estas informações são nome, SIAPE, departamento e cargo atual.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar algum cargo, sendo que há algum pré-selecionado.
Execução Normal	O funcionário seleciona o cargo desejado e a ordem de exibição dos docentes, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório.
Exceções	Se não existirem docentes referentes ao cargo selecionado, é emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.8 – Descrição do Caso de Uso Emitir Relatório por Departamento

Id Caso de Uso: 008	
Nome: Emitir Relatório por Departamento	
Descrição: Emite um relatório contendo informações dos docentes que trabalham no mesmo departamento. Estas informações são nome, SIAPE, curso e cargo.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar algum departamento, sendo que há algum pré-selecionado.
Execução Normal	O funcionário informa o departamento e a ordem de exibição dos docentes, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório.
Exceções	Se não existirem docentes referentes ao departamento selecionado, é emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.9 – Descrição do Caso de Uso Emitir Relatório por Titulação

Id Caso de Uso: 009	
Nome: Emitir Relatório por Titulação	
Descrição: Emite um relatório contendo informações dos docentes que possuem a mesma titulação. Estas informações são nome, SIAPE, departamento e cargo atual.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar alguma titulação, sendo que há alguma pré-selecionada.
Execução Normal	O funcionário informa a titulação desejada e a ordem de exibição dos docentes, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório.
Exceções	Se não existirem docentes referentes à titulação selecionada, é emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.10 – Descrição do Caso de Uso Emitir Relatório por Data de Admissão

Id Caso de Uso: 010	
Nome: Emitir Relatório por Data de Admissão	
Descrição: Emite um relatório contendo informações dos docentes, que foram admitidos em determinada data (mês e ano). Estas informações são nome, SIAPE, departamento e data de admissão.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar o mês e informar o ano.
Execução Normal	O funcionário informa a data (mês e ano) desejada.
Pós-Condições	O sistema emite o relatório ordenado por nome de docente.
Exceções	O relatório não é emitido se não houverem docentes admitidos na data previamente informada, sendo emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.11 – Descrição do Caso de Uso Emitir Relatório por Progressão do Mês

Id Caso de Uso: 011	
Nome: Emitir Relatório por Progressão do Mês	
Descrição: Emite um relatório contendo informações dos docentes, que terão progressão em determinada data (mês e ano). Estas informações são nome, SIAPE, departamento, última progressão e cargo atual.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar o mês e informar o ano.
Execução Normal	O funcionário informa a data (mês e ano) desejada e a ordem de exibição dos docentes, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório ordenado por nome de docente.
Exceções	O relatório não é emitido se não houver docentes que terão progressão na data previamente informada, sendo emitida uma mensagem indicando que o relatório é vazio.

6.3.3.2 Análise

O Modelo de Análise apresentado nesta seção é composto pelo Diagrama de Classes, o Dicionário de Atributos e Métodos e os seguintes Diagramas de Seqüência: Incluir Departamento, Alterar Departamento, Excluir Departamento, Incluir Docente, Alterar Docente, Excluir Docente, Consultar Docente, Modificar Progressão de Docente, Consultar Progressão de Docente, Modificar Regime de Docente, Consultar Regime de Docente, Emitir Relatório por Cargo, Emitir Relatório por Departamento, Emitir Relatório por Titulação, Emitir Relatório por Data de Admissão e Emitir Relatório por Progressão do Mês.

6.3.3.2.1 Diagrama de Classes

O Diagrama de Classes representa uma visão estática do sistema, porque a estrutura e o comportamento descritos são sempre válidos em qualquer ponto do ciclo de vida do produto de software. A Figura 6.3 ilustra o Diagrama de Classes do Produto de Software CPPD, para sua representação foram utilizados os elementos de modelagem sugeridos pela UML (BOOCH *et al*, 1999).

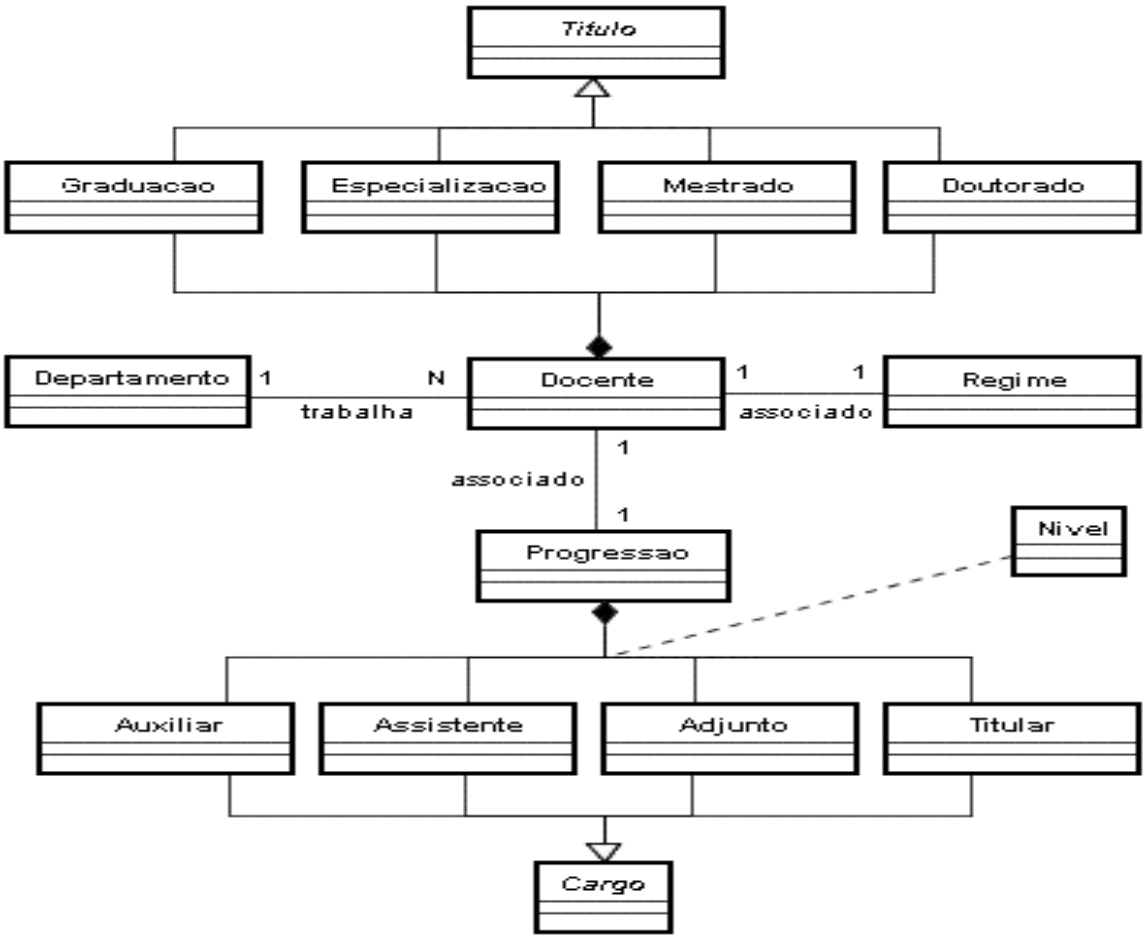


Figura 6.3 – Diagrama de Classes - Modelo de Análise

6.3.3.2.2 Dicionário de Atributos e Métodos

Nesta seção, é apresentado o Dicionário de Atributos e Métodos, que fornece uma descrição detalhada das classes. Essa descrição consiste em organizar os atributos e os métodos das classes, apresentando suas características. Além disso, contribui para não sobrecarregar visualmente o Diagrama de Classes. A seguir, é apresentado o Dicionário de Atributos e Métodos das seguintes classes:

- Departamento (Tabela 6.12);
- Título (Tabela 6.13);
- Doutorado (Tabela 6.14);
- Nivel (Tabela 6.15);
- Cargo (Tabela 6.16);
- Assistente (Tabela 6.17);
- Regime (Tabela 6.18);
- Progressão (Tabela 6.19);
- Docente (Tabela 6.20).

As classes Graduação, Especialização, Mestrado e Doutorado são semelhantes. As classes Mestrado e Doutorado diferem das classes Graduação e Especialização por

possuírem o atributo destinado a identificar a área de atuação e os métodos para atribuir e obter o seu valor. A classe Assistente é semelhante às classes Auxiliar, Adjunto e Titular.

Tabela 6.12 – Dicionário de Atributos e Métodos da Classe Departamento

Classe	**Departamento**					
Descrição	Identifica e manipula informações sobre o departamento onde os docentes exercem suas atividades.					
Atributos						
Identificador	sigla					
Descrição	Sigla do departamento					
Valores Possíveis	Qualquer seqüência de caracteres					
Tamanho	3	**Formato**	Letras		**TAD**	String
Identificador	descricao					
Descrição	Descrição do departamento					
Valores Possíveis	Qualquer seqüência de caracteres					
Tamanho	70	**Formato**	Letras		**TAD**	String
Métodos						
Identificador	criarDepartamento					
Descrição Funcional	Cria um objeto Departamento					
Parâmetros de Entrada	nenhum			**TAD**	nenhum	
Parâmetros de Saída	nenhum			**TAD**	nenhum	
Identificador	setSigla					
Descrição Funcional	Seta a sigla do departamento					
Parâmetros de Entrada	sigla			**TAD**	String	
Parâmetros de Saída	nenhum			**TAD**	nenhum	
Identificador	getSigla					
Descrição Funcional	Obtém a sigla do departamento					
Parâmetros de Entrada	nenhum			**TAD**	nenhum	
Parâmetros de Saída	sigla			**TAD**	String	
Identificador	setDescricao					
Descrição Funcional	Seta a descrição do departamento					
Parâmetros de Entrada	descricao			**TAD**	String	
Parâmetros de Saída	nenhum			**TAD**	nenhum	
Identificador	getDescricao					
Descrição Funcional	Obtém a descrição do departamento					
Parâmetros de Entrada	nenhum			**TAD**	nenhum	
Parâmetros de Saída	descricao			**TAD**	String	

Tabela 6.13 – Dicionário de Atributos e Métodos da Classe Título

Classe	**Titulo**				
Descrição	Identifica e manipula informações sobre a carreira profissional de docentes.				
Atributos					
Identificador	Curso				
Descrição	Curso realizado				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	50	**Formato**	Letras	**TAD**	String
Identificador	Data				
Descrição	Data de conclusão do curso				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	String
Identificador	Local				
Descrição	Local onde realizou o curso				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	50	**Formato**	Letras	**TAD**	String
Métodos					
Identificador	setCurso				
Descrição Funcional	Seta o curso realizado pelo docente				
Parâmetros de Entrada	Curso		**TAD**	String	
Parâmetros de Saída	Nenhum		**TAD**	nenhum	
Identificador	getCurso				
Descrição Funcional	Obtém o curso realizado pelo docente				
Parâmetros de Entrada	Nenhum		**TAD**	nenhum	
Parâmetros de Saída	Curso		**TAD**	String	
Identificador	setData				
Descrição Funcional	Seta a data de conclusão do curso				
Parâmetros de Entrada	data		**TAD**	String	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getData				
Descrição Funcional	Obtém a data de conclusão do curso				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	data		**TAD**	String	
Identificador	setLocal				
Descrição Funcional	Seta o local onde o curso foi realizado				
Parâmetros de Entrada	local		**TAD**	String	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getLocal				

Descrição Funcional	Obtém o local onde o curso foi realizado		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	local	**TAD**	String

Tabela 6.14 – Dicionário de Atributos e Métodos da Classe Doutorado

Classe	**Doutorado (herda as características da classe Titulo)**				
Descrição	Identifica e manipula as informações sobre o doutorado do docente.				
Atributos					
Identificador	area				
Descrição	Área em que foi realizado o doutorado				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	50	**Formato**	Letras	**TAD**	String
Métodos					
Identificador	criarDoutorado				
Descrição Funcional	Cria um objeto Doutorado				
Parâmetros de Entrada	nenhum			**TAD**	nenhum
Parâmetros de Saída	nenhum			**TAD**	nenhum
Identificador	setArea				
Descrição Funcional	Seta a área em que foi realizado o doutorado				
Parâmetros de Entrada	area			**TAD**	String
Parâmetros de Saída	nenhum			**TAD**	nenhum
Identificador	getArea				
Descrição Funcional	Obtém a área em que foi realizado o doutorado				
Parâmetros de Entrada	nenhum			**TAD**	nenhum
Parâmetros de Saída	area			**TAD**	String

Tabela 6.15 – Dicionário de Atributos e Métodos da Classe Nivel

Classe	**Nivel**				
Descrição	Identifica e manipula o nível de progressão de docentes.				
Atributos					
Identificador	data				
Descrição	Qualquer seqüência de caracteres				
Valores Possíveis	Datas				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	String
Identificador	portaria				
Descrição	Portaria do nível.				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	Letras e Números	**TAD**	String
Identificador	dataPortaria				
Descrição	Data da portaria do nível.				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	String

Métodos			
Identificador	criarNivel		
Descrição Funcional	Cria um objeto Nivel		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	setData		
Descrição Funcional	Seta a data do nível		
Parâmetros de Entrada	data	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getData		
Descrição Funcional	Obtém a data do nível		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	data	**TAD**	String
Identificador	setPortaria		
Descrição Funcional	Seta a portaria do nível		
Parâmetros de Entrada	portaria	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getPortaria		
Descrição Funcional	Obtém a portaria do nível		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	portaria	**TAD**	String
Identificador	setDataPortaria		
Descrição Funcional	Seta a data de portaria do nível		
Parâmetros de Entrada	dataPortaria	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getDataPortaria		
Descrição Funcional	Obtém a data de portaria do nível.		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	dataPortaria	**TAD**	String

Tabela 6.16 – Dicionário de Atributos e Métodos da Classe Cargo

Classe	**Cargo**
Descrição	Identifica e manipula informações sobre o cargo de docentes.
Atributos	
Identificador	niveis
Descrição	Niveis relacionados ao cargo do docente
TAD	Vector de Objetos Nivel
Métodos	
Identificador	addNivel
Descrição Funcional	Adiciona um nível ao cargo do docente

Parâmetros de Entrada	nivel	**TAD**	Objeto Nivel
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getNivel		
Descrição Funcional	Obtém determinado nível do cargo do docente		
Parâmetros de Entrada	numNivel	**TAD**	Inteiro
Parâmetros de Saída	nivel	**TAD**	Objeto Nivel
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.17 – Dicionário de Atributos e Métodos da Classe Assistente

Classe	**Assistente (herda as características da classe Cargo)**		
Descrição	Identifica e manipula as informações sobre o cargo Assistente do docente.		
Métodos			
Identificador	criarAssistente		
Descrição Funcional	Cria um objeto Assistente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.18 – Dicionário de Atributos e Métodos da Classe Regime

Classe	**Regime**				
Descrição	Identifica e manipula informações sobre o regime de docentes.				
Atributos					
Identificador	tipoRegime				
Descrição	Tipo de regime do docente				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	Duas Posições	**TAD**	String
Identificador	dataPortaria				
Descrição	Data da portaria do regime				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	String
Identificador	numeroPortaria				
Descrição	Número da portaria relacionada ao regime				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	Números	**TAD**	String
Métodos					
Identificador	criarRegime				
Descrição Funcional	Cria um objeto Regime				
Parâmetros de Entrada	nenhum			**TAD**	nenhum
Parâmetros de Saída	nenhum			**TAD**	nenhum
Identificador	setTipoRegime				
Descrição Funcional	Seta o tipo de regime do docente				
Parâmetros de Entrada	tipoRegime			**TAD**	String

Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getTipoRegime		
Descrição Funcional	Obtém o tipo de regime do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	tipoRegime	**TAD**	String
Identificador	setDataPortaria		
Descrição Funcional	Seta a data da portaria relacionada ao regime		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	data	**TAD**	String
Identificador	getDataPortaria		
Descrição Funcional	Obtém a data da portaria relacionada ao regime		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	dataPortaria	**TAD**	String
Identificador	setNumeroPortaria		
Descrição Funcional	Seta o número da portaria relacionada ao regime		
Parâmetros de Entrada	numeroPortaria	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getNumeroPortaria		
Descrição Funcional	Obtém o número da portaria relacionada ao regime		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	numeroPortaria	**TAD**	String

Tabela 6.19 – Dicionário de Atributos e Métodos da Classe Progressao

Classe	**Progressao**
Descrição	Identifica e manipula informações sobre a progressão de docentes.
Atributos	
Identificador	auxiliar
Descrição	Progressão relativa ao cargo auxiliar do docente
TAD	Objeto Auxiliar
Identificador	assistente
Descrição	Progressão relativa ao cargo assistente do docente
TAD	Objeto Assistente
Identificador	adjunto
Descrição	Progressão relativa ao cargo adjunto do docente
TAD	Objeto Adjunto
Identificador	titular
Descrição	Progressão relativa ao cargo titular do docente
TAD	Objeto Titular

Métodos			
Identificador	criarProgressao		
Descrição Funcional	Cria um objeto Progressao		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	setAuxiliar		
Descrição Funcional	Seta os dados do cargo auxiliar relativo à progressão		
Parâmetros de Entrada	auxiliar	**TAD**	Objeto Auxiliar
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getAuxiliar		
Descrição Funcional	Obtém os dados do cargo auxiliar relativo à progressão		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	auxiliar	**TAD**	Objeto Auxiliar
Identificador	setAssistente		
Descrição Funcional	Seta os dados do cargo assistente relativo à progressão		
Parâmetros de Entrada	assistente	**TAD**	Objeto Assistente
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getAssistente		
Descrição Funcional	Obtém os dados do cargo assistente relativo à progressão		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	assistente	**TAD**	Objeto Assistente
Identificador	setAdjunto		
Descrição Funcional	Seta os dados do cargo adjunto relativo à progressão		
Parâmetros de Entrada	adjunto	**TAD**	Objeto Adjunto
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getAdjunto		
Descrição Funcional	Obtém os dados do cargo adjunto relativo à progressão		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	adjunto	**TAD**	Objeto Adjunto
Identificador	setTitular		
Descrição Funcional	Seta os dados do cargo titular relativo à progressão		
Parâmetros de Entrada	titular	**TAD**	Objeto Titular
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getTitular		
Descrição Funcional	Obtém os dados do cargo titular relativo à progressão		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	titular	**TAD**	Objeto Titular

Tabela 6.20 – Dicionário de Atributos e Métodos da Classe Docente

Classe	**Docente**					
Descrição	Identifica e manipula informações sobre os docentes.					
Atributos						
Identificador	siape					
Descrição	Número siape do docente					
Valores Possíveis	1 a 2147483647					
Tamanho	10	**Formato**	Número	**TAD**	String	
Identificador	nome					
Descrição	Nome do docente					
Valores Possíveis	Qualquer seqüência de caracteres					
Tamanho	50	**Formato**	Letras	**TAD**	String	
Identificador	dataAdmissao					
Descrição	Data de admissão do docente na universidade					
Valores Possíveis	Qualquer seqüência de caracteres					
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	String	
Identificador	graduacao					
Descrição	Dados relativos à titulação de graduação do docente					
TAD	Objeto Graduacao					
Identificador	especializacao					
Descrição	Dados relativos à titulação de especialização do docente					
TAD	Objeto Especializacao					
Identificador	mestrado					
Descrição	Dados relativos à titulação de mestrado do docente					
TAD	Objeto Mestrado					
Identificador	doutorado					
Descrição	Dados relativos à titulação de doutorado do docente					
TAD	Objeto Doutorado					
Métodos						
Identificador	criarDocente					
Descrição Funcional	Cria um objeto Docente					
Parâmetros de Entrada	nenhum	**TAD**	nenhum			
Parâmetros de Saída	nenhum	**TAD**	nenhum			
Identificador	setSiape					
Descrição Funcional	Atribui o número siape do docente					
Parâmetros de Entrada	siape	**TAD**	Inteiro			
Parâmetros de Saída	nenhum	**TAD**	nenhum			
Identificador	getSiape					

Descrição Funcional	Obtém o número siape do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	siape	**TAD**	Inteiro
Identificador	setNome		
Descrição Funcional	Seta o nome do docente		
Parâmetros de Entrada	nome	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getNome		
Descrição Funcional	Obtém o nome do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nome	**TAD**	String
Identificador	setDataAdmissao		
Descrição Funcional	Seta a data de admissão do docente		
Parâmetros de Entrada	dataAdmissao	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getDataAdmissao		
Descrição Funcional	Obtém a data de admissão do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	dataAdmissao	**TAD**	String
Identificador	setGraduacao		
Descrição Funcional	Seta os dados relativos à titulação de graduação do docente		
Parâmetros de Entrada	graduacao	**TAD**	Objeto Graduacao
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getGraduacao		
Descrição Funcional	Obtém os dados relativos à titulação de graduação do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	graduacao	**TAD**	Objeto Graduacao
Identificador	setEspecializacao		
Descrição Funcional	Seta os dados relativos à titulação de especialização do docente		
Parâmetros de Entrada	especilizacao	**TAD**	Objeto Especializacao
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getEspecializacao		
Descrição Funcional	Obtém os dados relativos à titulação de especialização do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	especilizacao	**TAD**	Objeto Especializacao
Identificador	setMestrado		

Descrição Funcional	Seta os dados relativos à titulação de mestrado do docente		
Parâmetros de Entrada	mestrado	**TAD**	Objeto Mestrado
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getMestrado		
Descrição Funcional	Obtém os dados relativos à titulação de mestrado do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	mestrado	**TAD**	Objeto Mestrado
Identificador	setDoutorado		
Descrição Funcional	Seta os dados relativos à titulação de doutorado do docente		
Parâmetros de Entrada	doutorado	**TAD**	Objeto Doutorado
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getDoutorado		
Descrição Funcional	Obtém os dados relativos à titulação de doutorado do docente		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	doutorado	**TAD**	Objeto Doutorado

6.3.3.2.3 Diagramas de Seqüência

Um Diagrama de Seqüência descreve a maneira como os grupos de objetos colaboram em algum comportamento ao longo do tempo. Ele registra o comportamento de um único caso de uso, exibindo os objetos e as mensagens trocadas entre eles. A seguir, são apresentados os seguintes Diagramas de Seqüência:

- Incluir Departamento (Figura 6.4);
- Alterar Departamento (Figura 6.5);
- Excluir Departamento (Figura 6.6);
- Incluir Docente (Figura 6.7);
- Alterar Docente (Figura 6.8);
- Excluir Docente (Figura 6.9);
- Consultar Docente (Figura 6.10);
- Modificar Progressão de Docente (Figura 6.11);
- Consultar Progressão de Docente (Figura 6.12);
- Emitir Relatório por Cargo (Figura 6.13).

Os Diagramas de Seqüência do caso se uso Modificar Progressão de Docente e do caso se uso Modificar Regime de Docente são semelhantes. Os Diagramas de Seqüência do caso se uso Consultar Progressão de Docente e do caso se uso Consultar Regime de Docente são semelhantes. Os Diagramas de Seqüência para os casos de uso de emissão de relatórios são semelhantes.

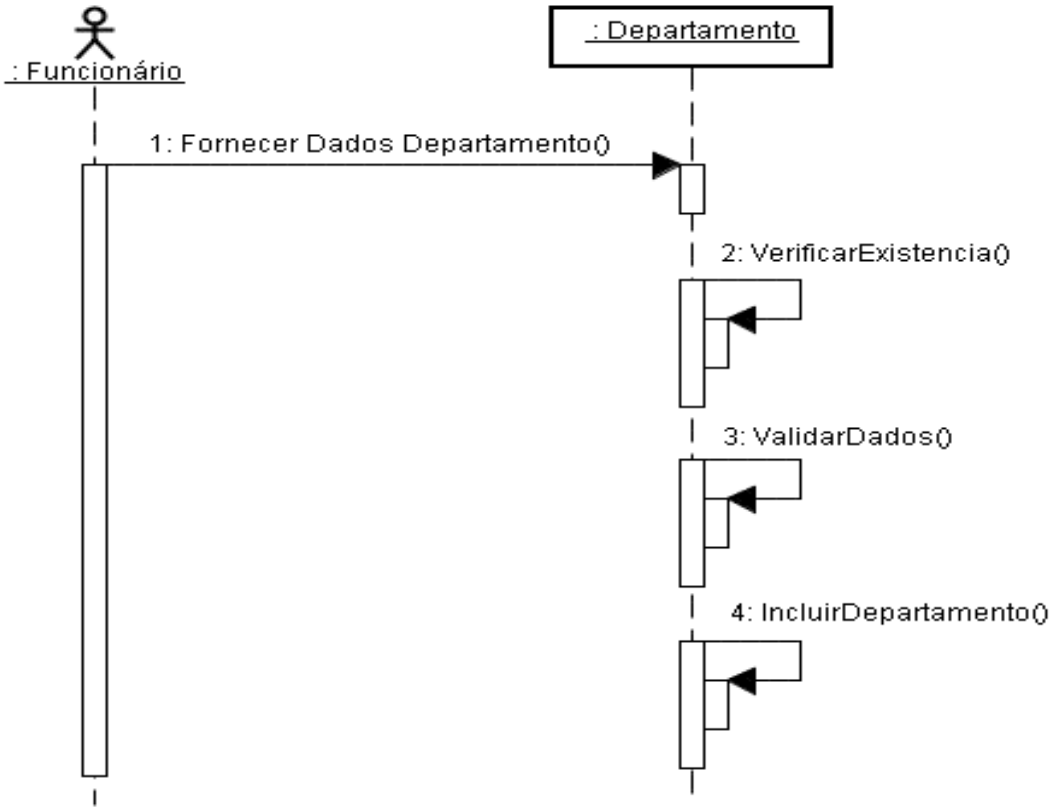


Figura 6.4 – Diagrama de Seqüência – Incluir Departamento

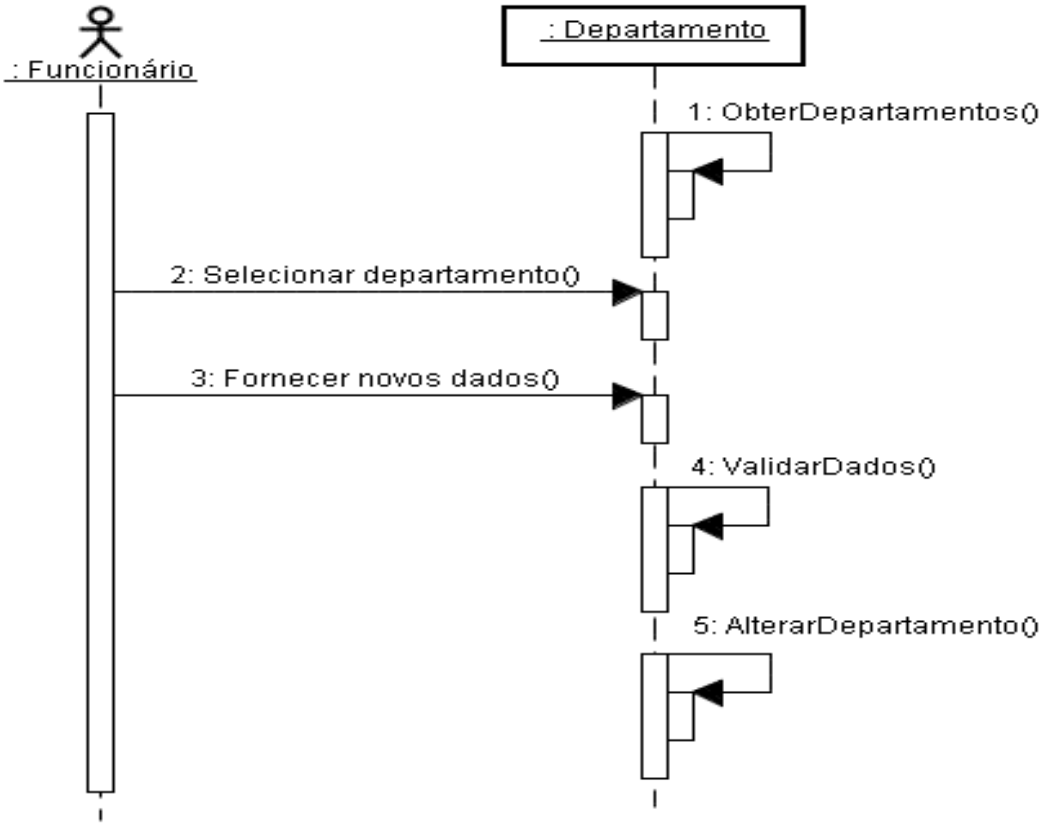


Figura 6.5 – Diagrama de Seqüência – Alterar Departamento

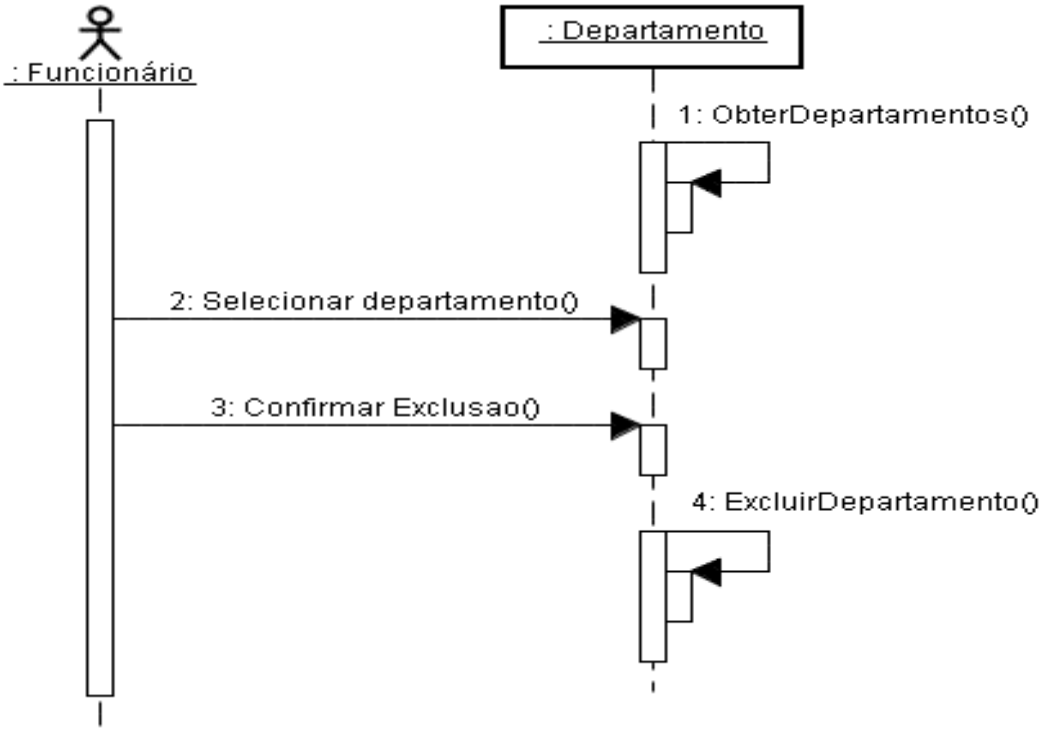


Figura 6.6 – Diagrama de Seqüência – Excluir Departamento

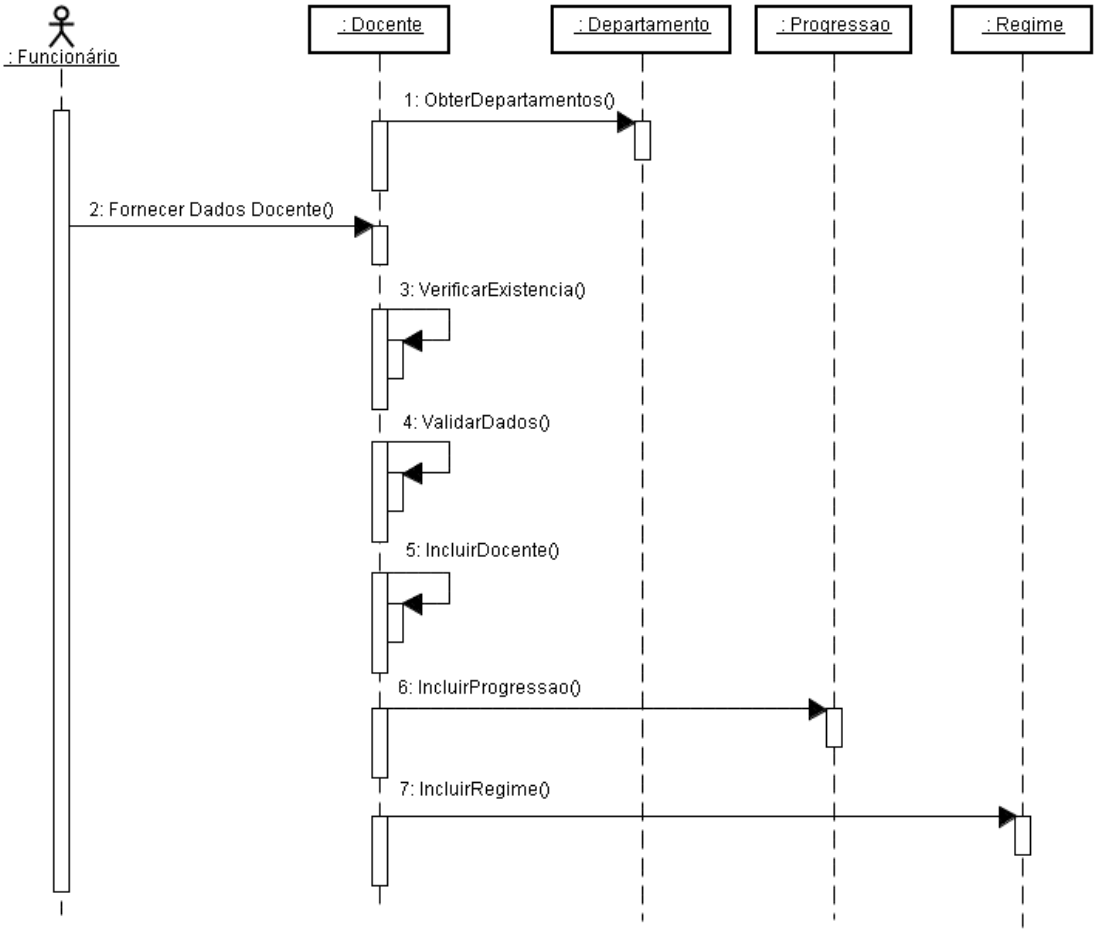


Figura 6.7 – Diagrama de Seqüência – Incluir Docente

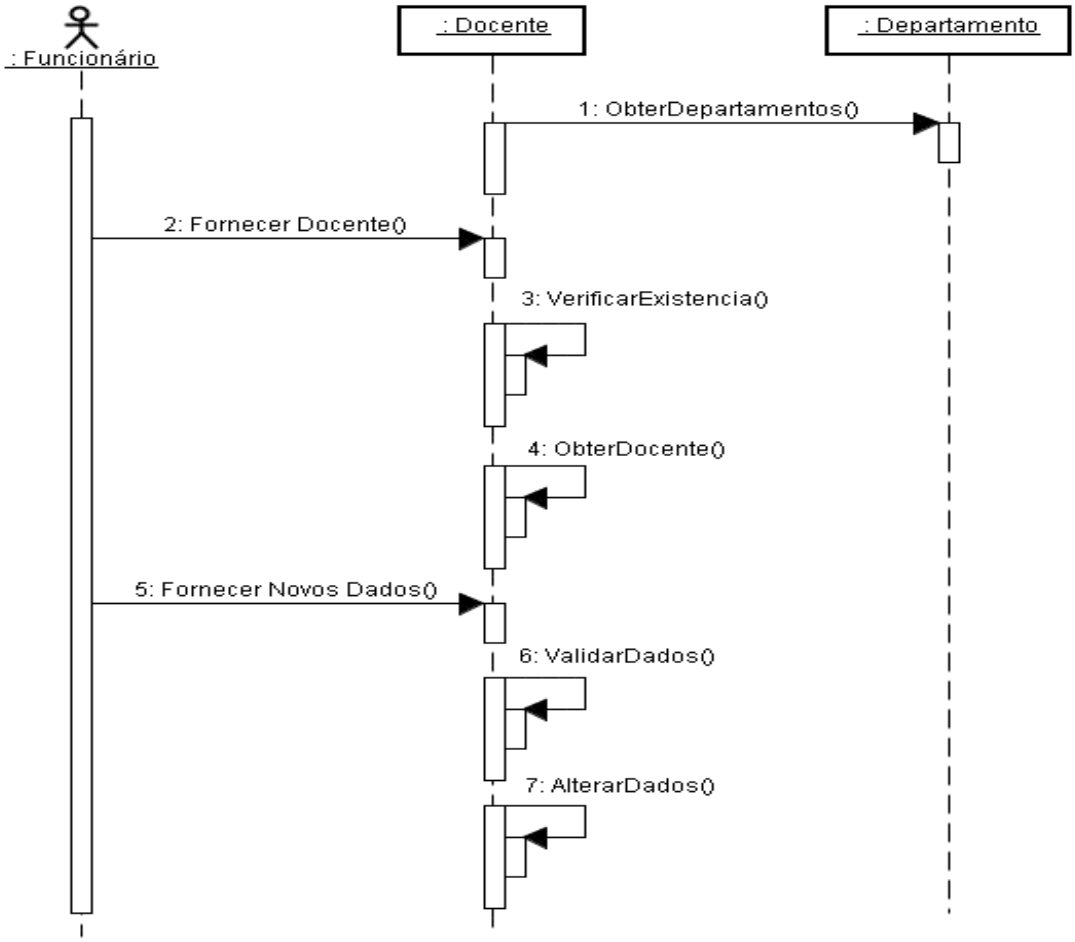


Figura 6.8 – Diagrama de Seqüência – Alterar Docente

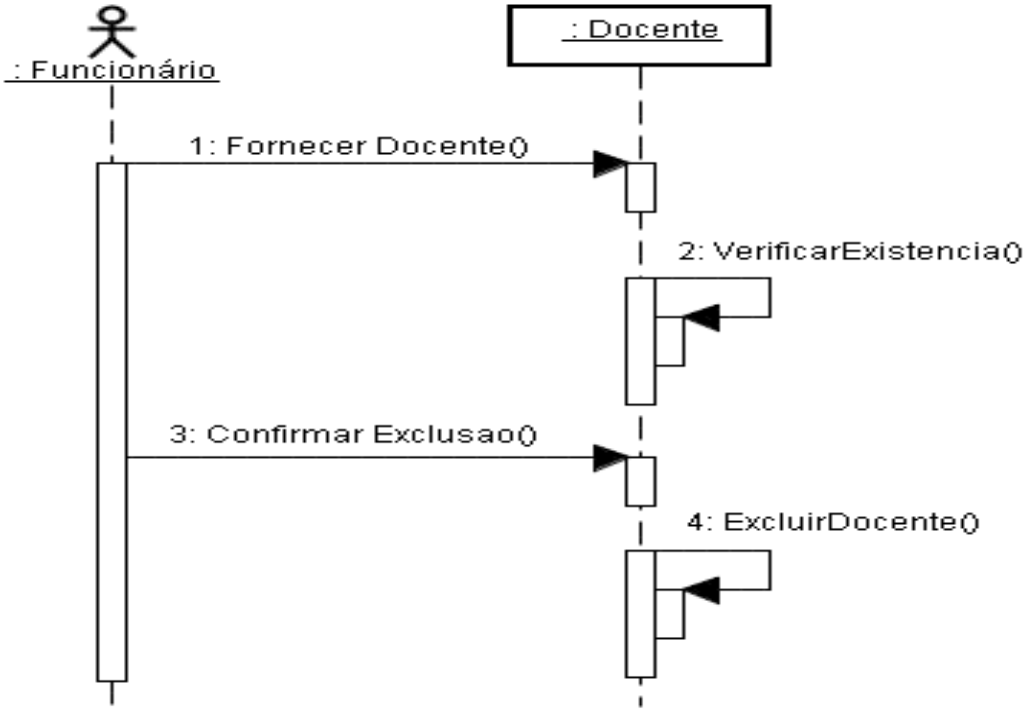


Figura 6.9 – Diagrama de Seqüência – Excluir Docente

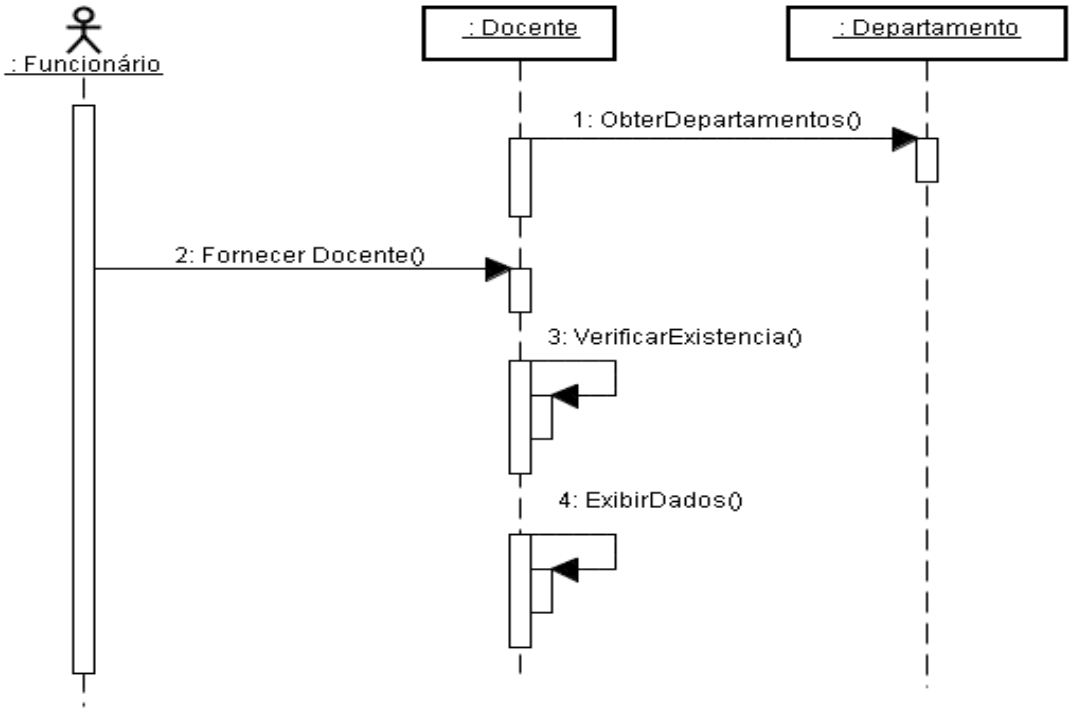


Figura 6.10 – Diagrama de Seqüência – Consultar Docente

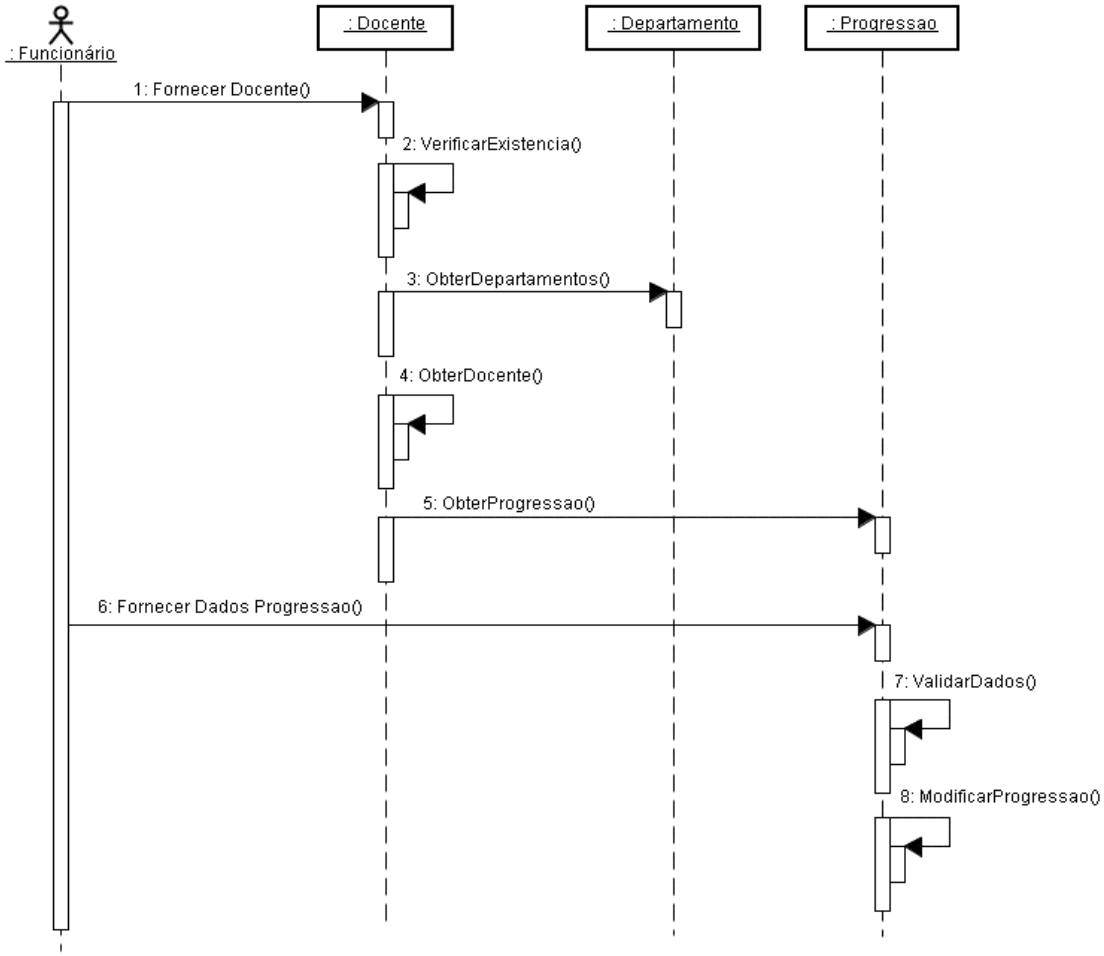


Figura 6.11 – Diagrama de Seqüência – Modificar Progressão de Docente

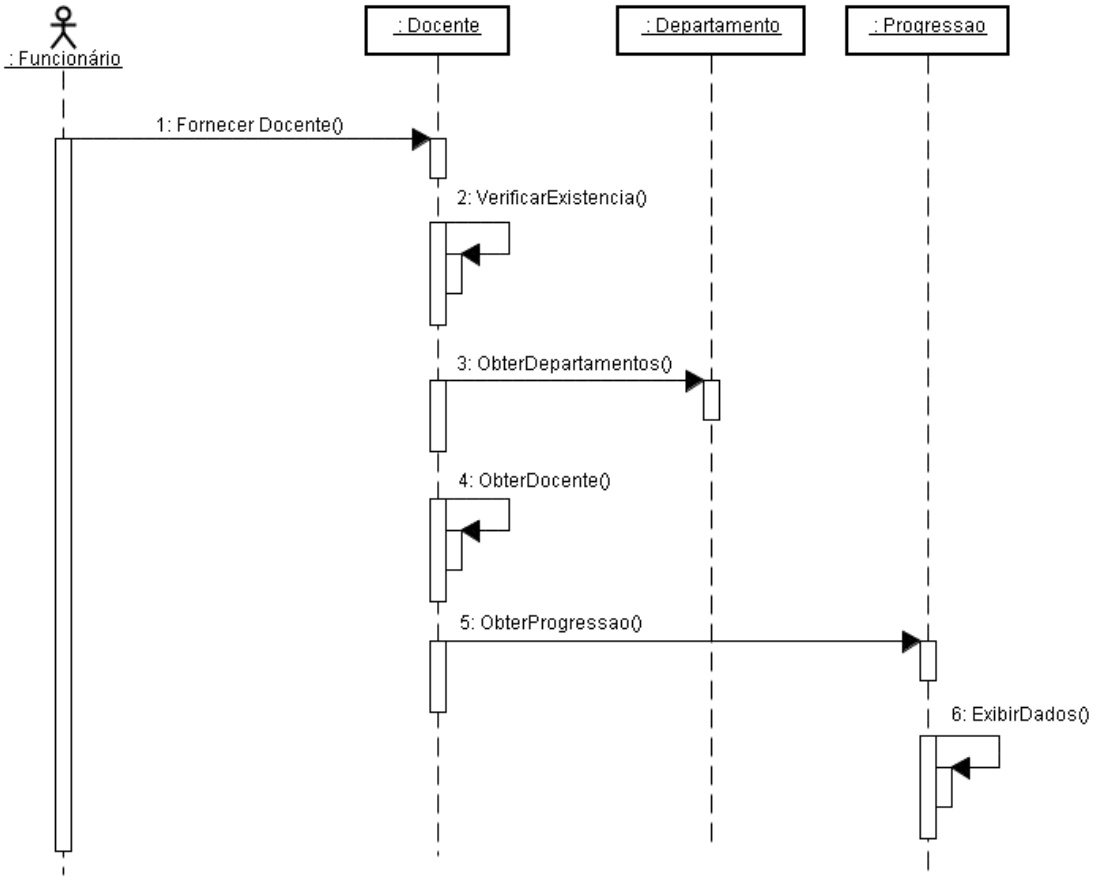


Figura 6.12 – Diagrama de Seqüência – Consultar Progressão de Docente

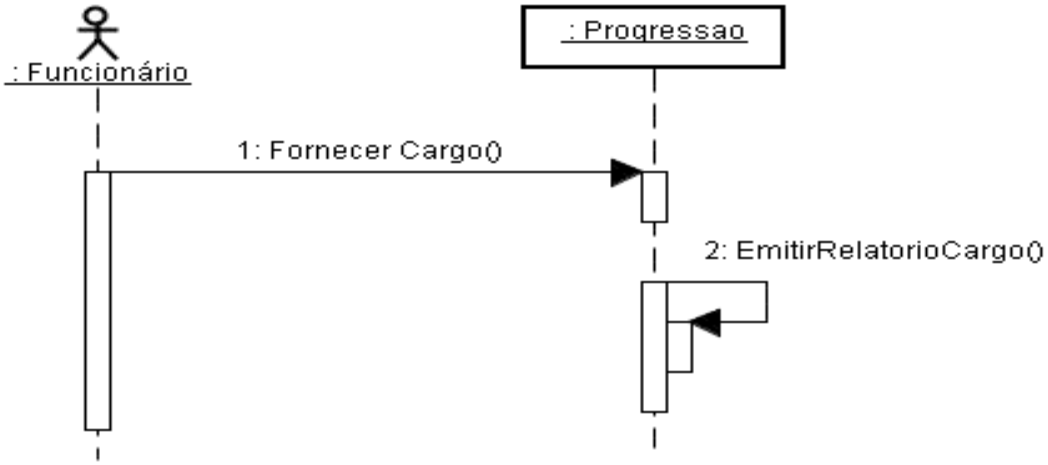


Figura 6.13 – Diagrama de Seqüência – Emitir Relatório por Cargo

6.3.3.3 Projeto

O Modelo de Projeto no contexto do trabalho é composto pelos mesmos diagramas do Modelo de Análise, porém engloba características de implementação (por exemplo, a interface do usuário e de periféricos, gerenciamento de banco de dados e comunicação com outros sistemas).

6.3.3.3.1 Diagrama de Classes

A Figura 6.14 ilustra o Diagrama de Classes do Produto de Software CPPD.

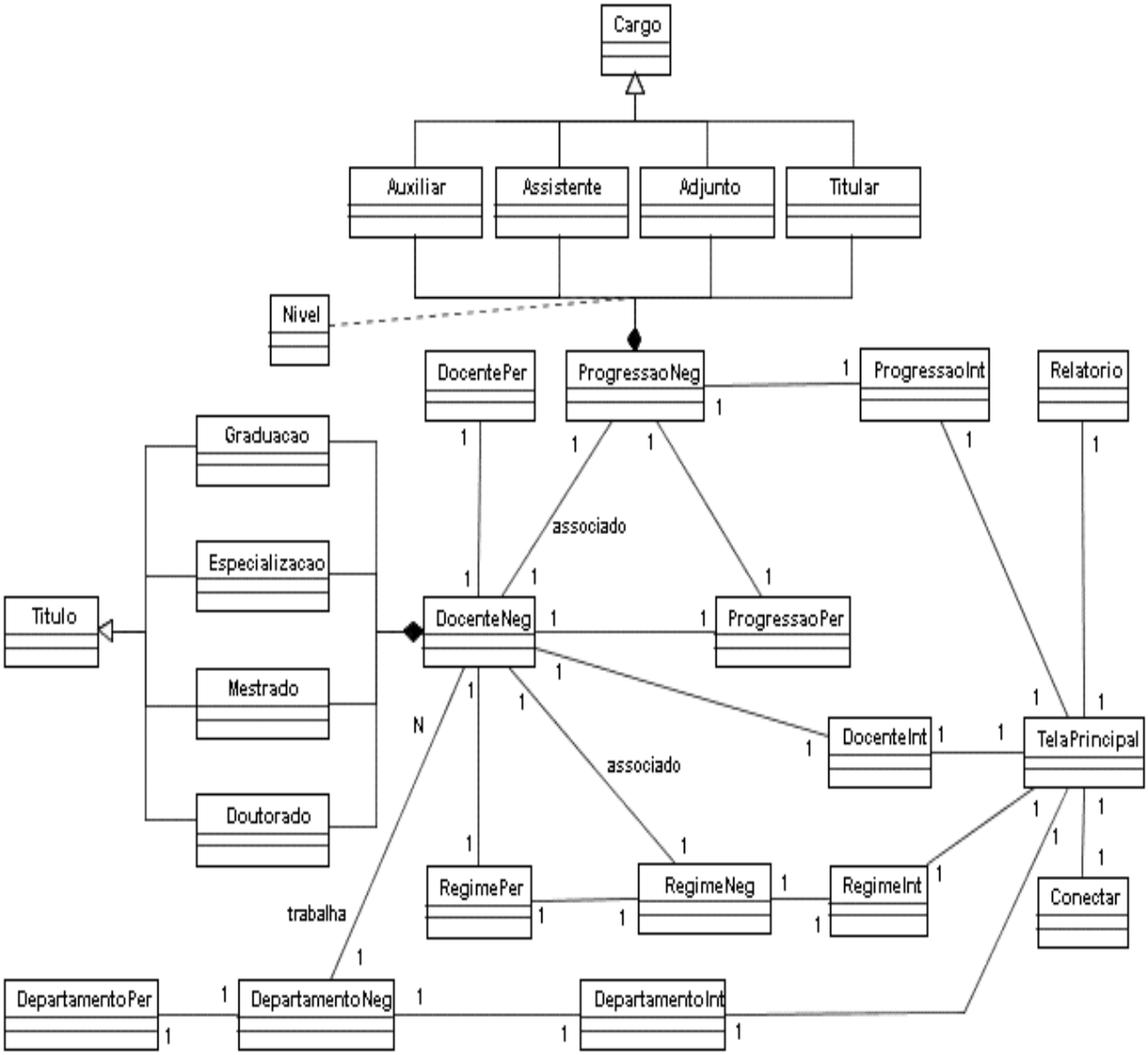


Figura 6.14 – Diagrama de Classes – Modelo de Projeto.

6.3.3.3.2 Dicionário de Atributos e Métodos

Algumas das classes presentes no Diagrama de Classes do Modelo de Projeto não existem do Dicionário de Atributos e Métodos do Modelo de Análise, assim como há algumas classes idênticas e outras parcialmente idênticas. As tabelas apresentam o Dicionário de Atributos e Métodos das seguintes classes que sofreram alguma alteração ou que foram acrescentados no Modelo de Projeto:

- Tela Principal (Tabela 6.21);
- Conectar (Tabela 6.22);
- Relatorio (Tabela 6.23);
- DocenteInt (Tabela 6.24);

- DocenteNeg (Tabela 6.25);
- DocentePer (Tabela 6.26);
- ProgressaoInt (Tabela 6.27);
- ProgressaoNeg (Tabela 6.28);
- ProgressaoPer (Tabela 6.29).

As classes Departamento e Docente do Modelo de Projeto são semelhantes, pois elas possuem operações de incluir, alterar e excluir, além de atributos referentes ao banco de dados. Sendo assim, são apresentados os Dicionários de Atributos e Métodos das classes relativas à classe Docente.

A classe DocenteNeg é semelhante à classe Docente, diferenciando-se por possuir o atributo departamento (utilizado como chave estrangeira no banco de dados e identificando o departamento relativo ao docente), os métodos para atribuir e obter o valor do atributo e os métodos referentes à comunicação com a camada de persistência.

As classes referentes a Progressao e Regime do Modelo de Projeto são semelhantes. Sendo assim, são apresentados os Dicionários de Atributos e Métodos das classes relativas à classe Progressao.

Assim como a classe DocenteNeg, a classe ProgressaoNeg é semelhante à classe Progressão, diferenciando-se por possuir os métodos que comunicam com a camada de persistência.

As classes Titulo, Graduação, Especialização, Mestrado, Doutorado, Nivel, Cargo, Auxiliar, Assistente, Adjunto e Titular são idênticas às do Modelo de Análise, portanto seus respectivos dicionários são os mesmos apresentados no Modelo de Análise.

Tabela 6.21 – Dicionário de Atributos e Métodos da Classe Tela Principal

Classe	**TelaPrincipal**				
Descrição	Tela principal da aplicação. Fornece acesso aos menus.				
Atributos					
Identificador	Siape				
Descrição	Número siape do docente				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Identificador	nome				
Descrição	Nome do docente				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	50	**Formato**	Letras	**TAD**	String

Identificador	flag				
Descrição	Indica se o número siape ou nome do técnico administrativo fornecido é válido				
Valores Possíveis	Verdadeiro ou Falso				
Tamanho	-	**Formato**	Verdadeiro ou Falso	**TAD**	Booleano

Tabela 6.22 – Dicionário de Atributos e Métodos da Classe Conectar

Classe	**Conectar**		
Descrição	Responsável por prover acesso ao banco de dados.		
Métodos			
Identificador	getCon		
Descrição Funcional	Retorna a conexão com o servidor		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	conn	**TAD**	Objeto Connection

Tabela 6.23 – Dicionário de Atributos e Métodos da Classe Relatorio

Classe	**Relatorio**		
Descrição	Emite relatórios ao funcionário.		
Métodos			
Identificador	EmitirRelatorio		
Descrição Funcional	Gera os relatórios		
Parâmetros de Entrada	query, relatorio	**TAD**	String, Arquivo
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.24 – Dicionário de Atributos e Métodos da Classe DocenteInt

Classe	**DocenteInt**				
Descrição	Responsável pela troca de mensagem entre o sistema e o usuário em relação às operações sobre os Docentes.				
Atributos					
Identificador	siape				
Descrição	número siape do docente				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Identificador	docente				
Descrição	Um docente do sistema				
TAD	Objeto DocenteNeg				
Métodos					
Identificador	ValidarDados				
Descrição Funcional	Verifica se os dados foram preenchidos corretamente				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	retorno		**TAD**	Booleano	

Tabela 6.25 – Dicionário de Atributos e Métodos da Classe DocenteNeg

Classe	**DocenteNeg**					
Descrição	Identifica e manipula informações sobre os docentes.					
Atributos						
Identificador	departamento					
Descrição	Código do departamento onde o docente realiza suas atividades					
Valores Possíveis	-2147483648 a 2147483647					
Tamanho	10	**Formato**	Números		**TAD**	Inteiro
Métodos						
Identificador	setDepartamento					
Descrição Funcional	Seta o código do departamento onde o docente trabalha					
Parâmetros de Entrada	departamento			**TAD**	Inteiro	
Parâmetros de Saída	nenhum			**TAD**	nenhum	
Identificador	getDepartamento					
Descrição Funcional	Obtém o código do departamento onde o docente trabalha					
Parâmetros de Entrada	nenhum			**TAD**	nenhum	
Parâmetros de Saída	departamento			**TAD**	Inteiro	
Identificador	VerificarExistencia					
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, verificar se determinado docente existe.					
Parâmetros de Entrada	siape			**TAD**	Inteiro	
Parâmetros de Saída	retorno			**TAD**	Booleano	
Identificador	VerificarExistencia					
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, verificar se determinado docente existe.					
Parâmetros de Entrada	nome			**TAD**	String	
Parâmetros de Saída	retorno			**TAD**	Booleano	
Identificador	ObterDocente					
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, obter um docente no banco de dados.					
Parâmetros de Entrada	siape			**TAD**	Inteiro	
Parâmetros de Saída	docente			**TAD**	Objeto DocenteNeg	
Identificador	AlterarDocente					
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, alterar os dados de determinado docente no Banco de Dados.					
Parâmetros de Entrada	docente			**TAD**	Objeto DocenteNeg	
Parâmetros de Saída	nenhum			**TAD**	nenhum	
Identificador	ExcluirDocente					
Descrição Funcional	Comunica com a camada de persistência para,					

	posteriormente, excluir um docente no banco de dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.26 – Dicionário de Atributos e Métodos da Classe DocentePer

Classe	**DocentePer (classe estática)**		
Descrição	Responsável pelas operações sobre o Banco de Dados.		
Métodos			
Identificador	VerificarExistencia		
Descrição Funcional	Verifica se determinado docente existe no Banco de Dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	retorno	**TAD**	Booleano
Identificador	VerificarExistencia		
Descrição Funcional	Verifica se determinado docente existe no Banco de Dados		
Parâmetros de Entrada	nome	**TAD**	String
Parâmetros de Saída	retorno	**TAD**	Booleano
Identificador	IncluirDocente		
Descrição Funcional	Inclui um docente no Banco de Dados		
Parâmetros de Entrada	docente	**TAD**	Objeto DocenteNeg
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	AlterarDocente		
Descrição Funcional	Altera os dados de determinado docente no Banco de Dados		
Parâmetros de Entrada	docente	**TAD**	Objeto DocenteNeg
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ExcluirDocente		
Descrição Funcional	Exclui um docente do Banco de Dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ObterDocente		
Descrição Funcional	Obtém determinado docente do banco de dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	docente	**TAD**	Objeto DocenteNeg
Identificador	ModificarCargo		
Descrição Funcional	Altera o cargo atual do docente no banco de dados		
Parâmetros de Entrada	p, siape	**TAD**	Objeto ProgressaoNeg, Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.27 – Dicionário de Atributos e Métodos da Classe ProgressaoInt

Classe	**ProgressaoInt**					
Descrição	Responsável pela troca de mensagem entre o sistema e o usuário em relação às operações sobre as progressões dos docentes.					
Atributos						
Identificador	siape					
Descrição	Número siape do docente					
Valores Possíveis	1 a 2147483647					
Tamanho	10	**Formato**	Números		**TAD**	Inteiro
Identificador	progressao					
Descrição	Progressão de determinado docente					
TAD	Objeto ProgressaoNeg					
Métodos						
Identificador	ValidarDados					
Descrição Funcional	Verifica se os dados foram preenchidos corretamente					
Parâmetros de Entrada	nenhum			**TAD**	nenhum	
Parâmetros de Saída	retorno			**TAD**	Booleano	

Tabela 6.28 – Dicionário de Atributos e Métodos da Classe ProgressaoNeg

Classe	**ProgressaoNeg**		
Descrição	Identifica e manipula informações sobre a progressão de docentes.		
Métodos			
Identificador	ModificarProgressao		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, alterar os dados de determinada progressão no banco de dados.		
Parâmetros de Entrada	p, siape	**TAD**	Objeto ProgressaoNeg, Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ObterProgressao		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, obter determinada progressão no banco de dados.		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	progressao	**TAD**	Objeto ProgressaoNeg

Tabela 6.29 – Dicionário de Atributos e Métodos da Classe ProgressaoPer

Classe	**ProgressaoPer (classe estática)**		
Descrição	Responsável pelas operações sobre o Banco de Dados.		
Métodos			
Identificador	CriarProgressao		
Descrição Funcional	Insere uma progressão no banco de dados (em branco)		
Parâmetros de Entrada	siape	**TAD**	Inteiro

Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ModificarProgressao		
Descrição Funcional	Altera os dados de determinada progressão no banco de dados		
Parâmetros de Entrada	p, siape	**TAD**	Objeto ProgressaoNeg, Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ObterProgressao		
Descrição Funcional	Obtém determinada progressão no Banco de Dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	progressao	**TAD**	Objeto ProgressaoNeg

6.3.3.3.3 Diagramas de Seqüência

A seguir, são apresentados os seguintes Diagramas de Seqüência:

- Incluir Departamento (Figura 6.15);
- Alterar Departamento (Figura 6.16);
- Excluir Departamento (Figura 6.17);
- Excluir Docente (Figura 6.18);
- Incluir Docente (Figura 6.19);
- Alterar Docente (Figura 6.20);
- Consultar Docente (Figura 6.21);
- Modificar Progressão de Docente (Figura 6.22);
- Consultar Progressão de Docente (Figura 6.23);
- Emitir Relatório por Cargo (Figura 6.24).

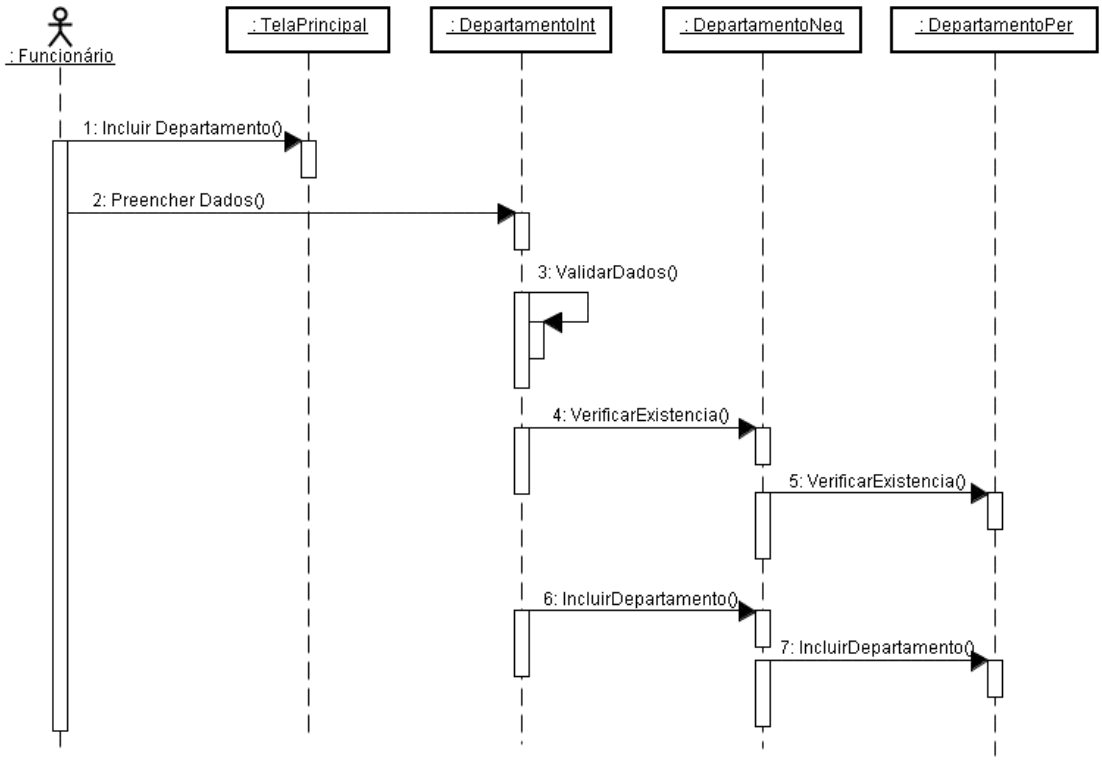


Figura 6.15 – Diagrama de Seqüência – Incluir Departamento

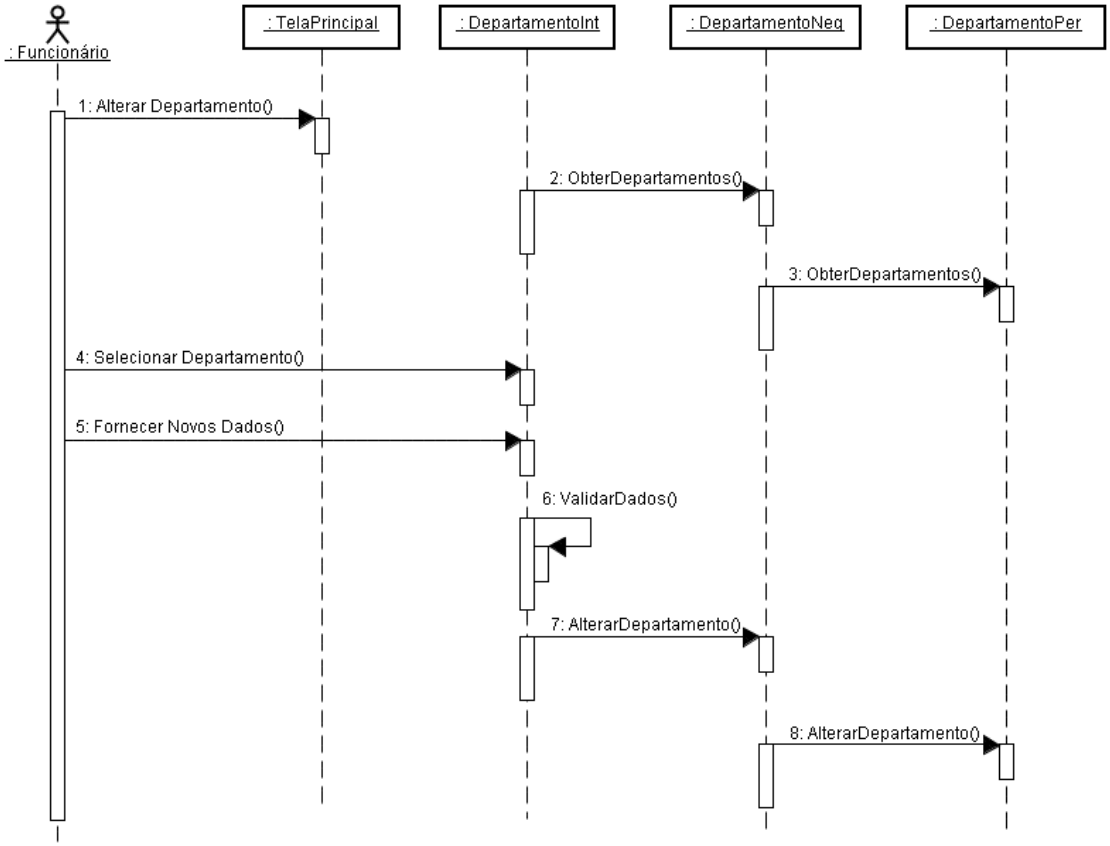


Figura 6.16 – Diagrama de Seqüência – Alterar Departamento

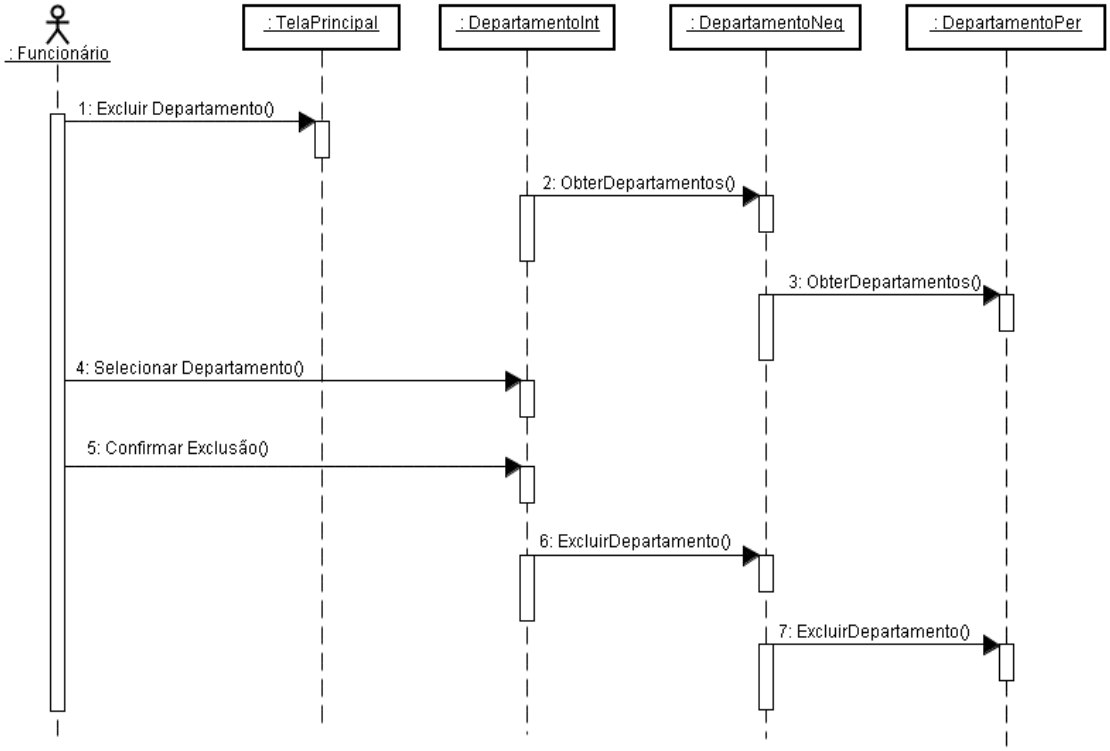


Figura 6.17 – Diagrama de Seqüência – Excluir Departamento

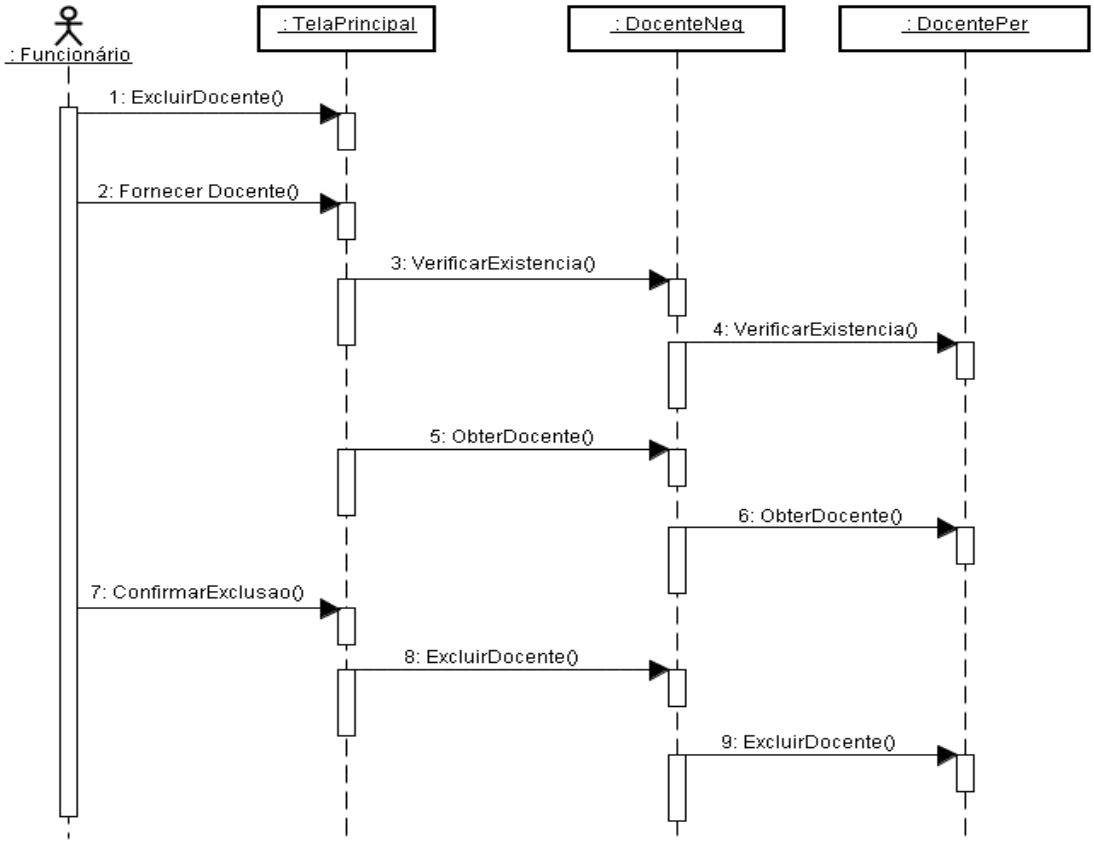


Figura 6.18 – Diagrama de Seqüência – Excluir Docente

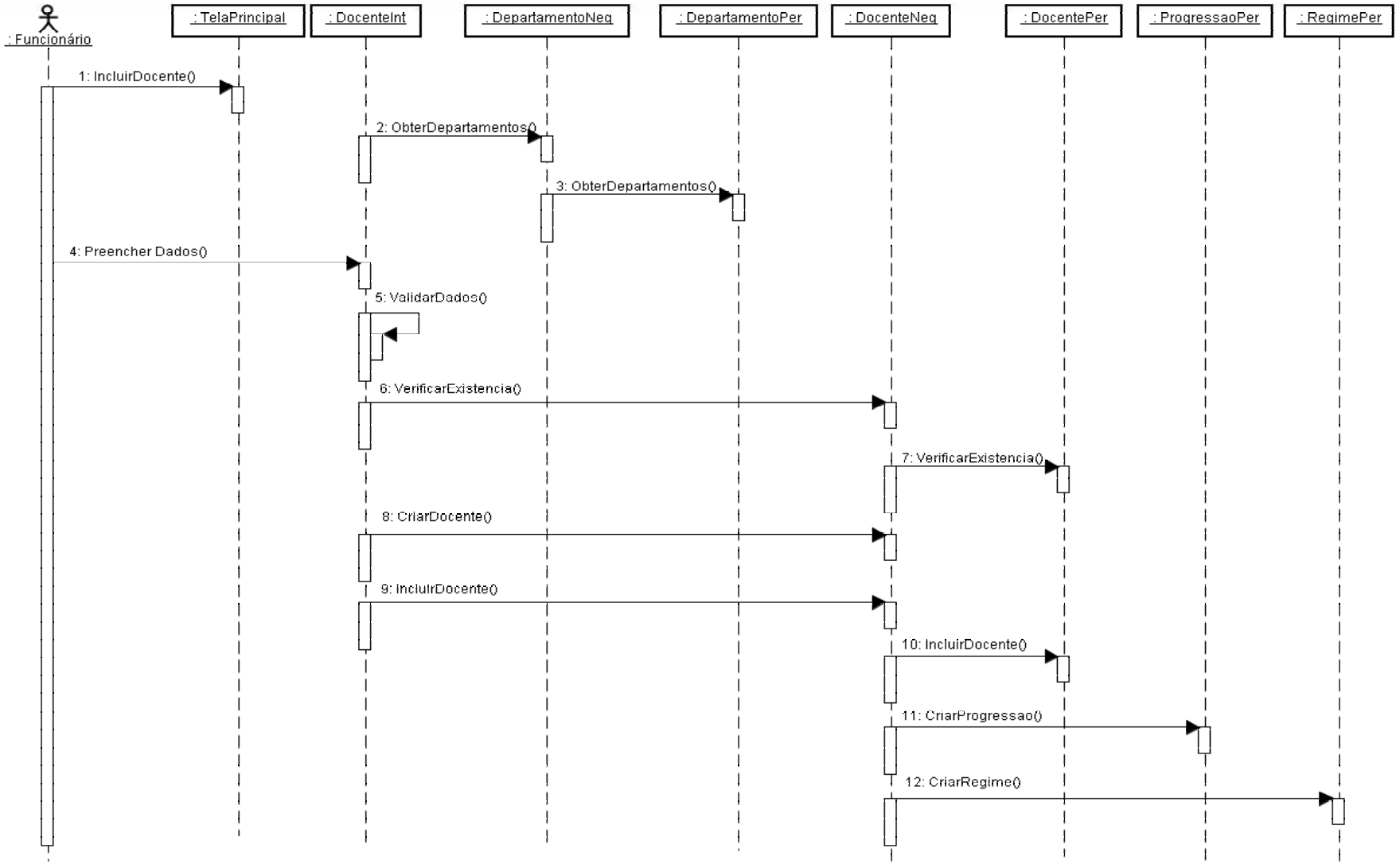


Figura 6.19 – Diagrama de Seqüência – Incluir Docente

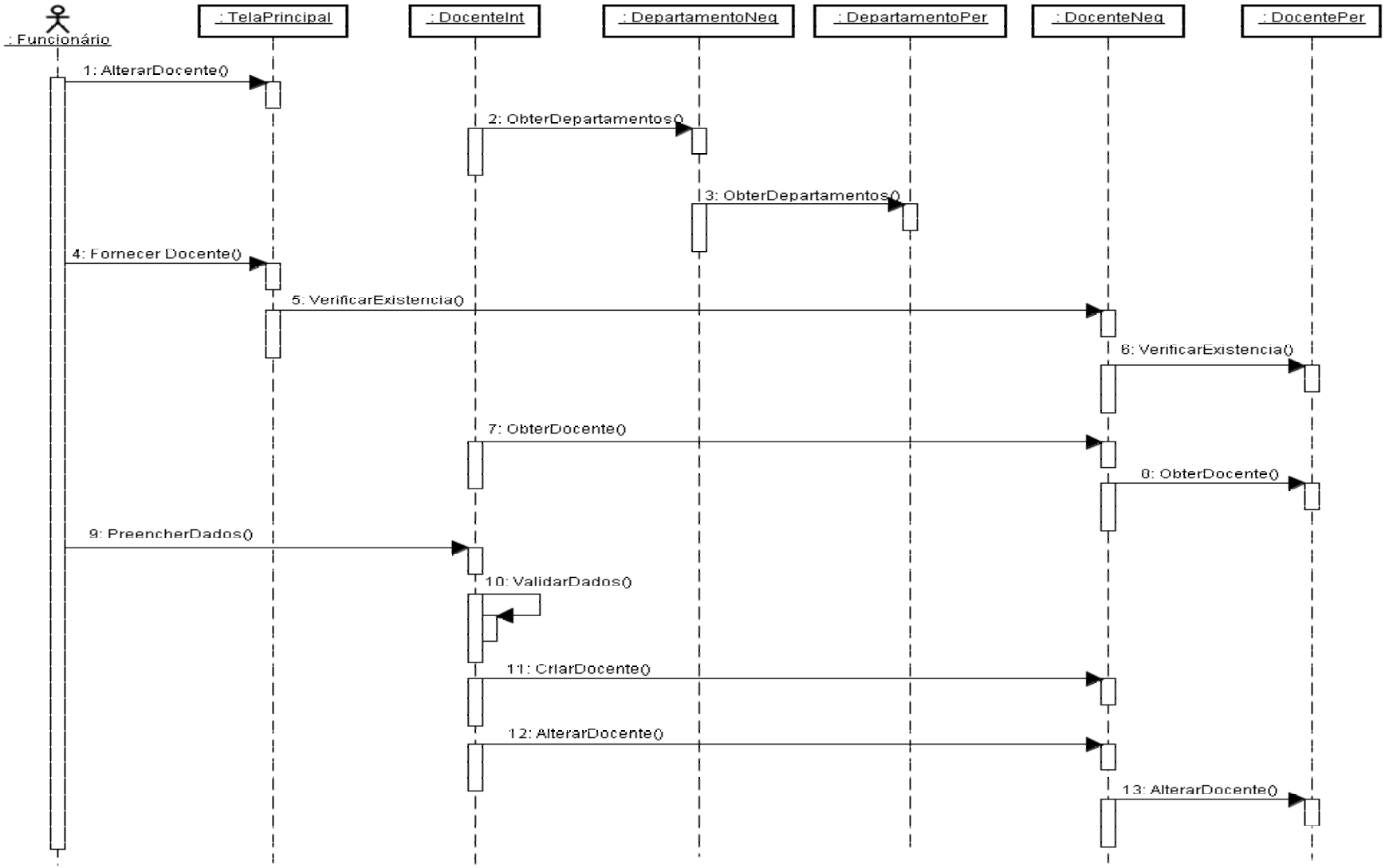


Figura 6.20 – Diagrama de Seqüência – Alterar Docente

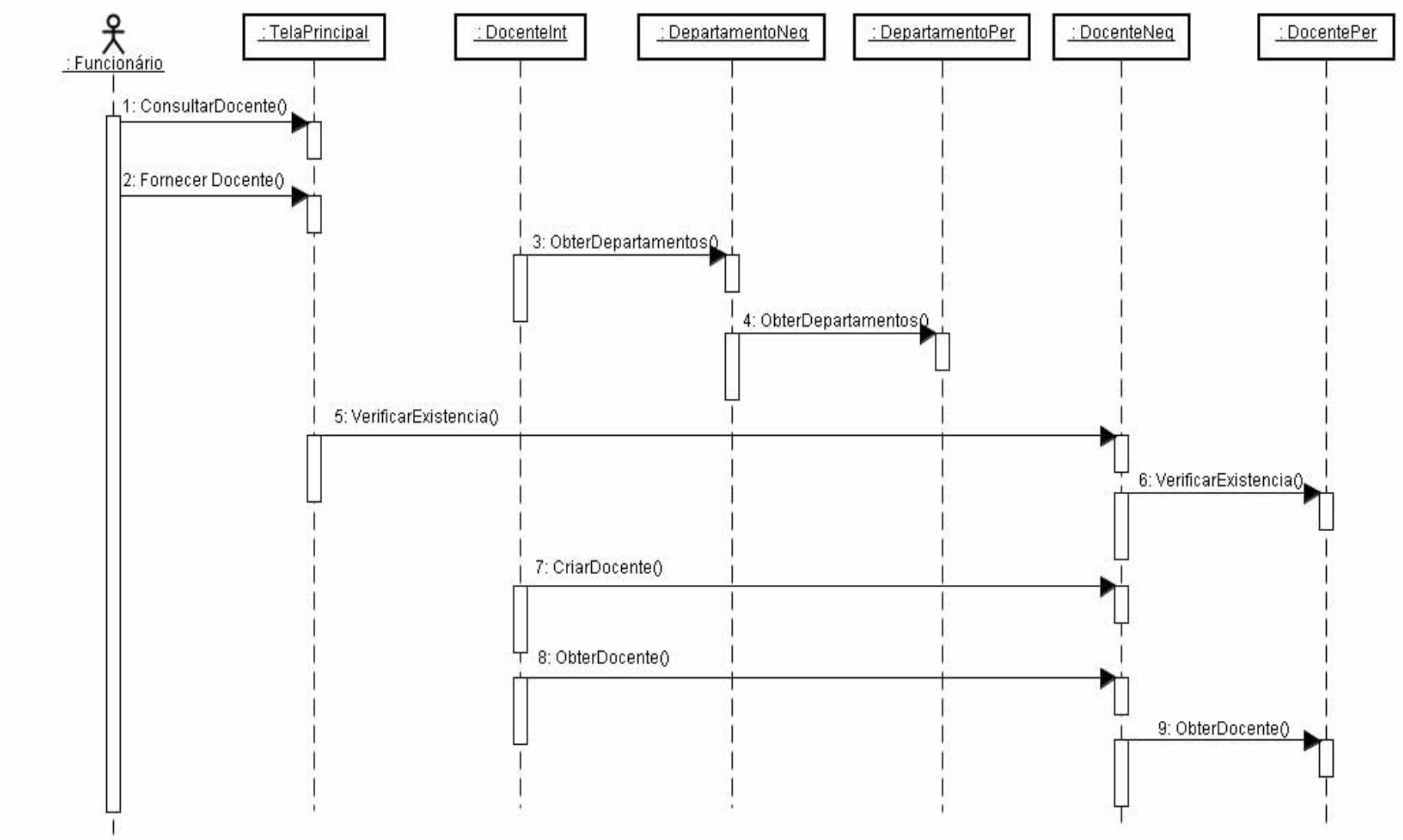


Figura 6.21 – Diagrama de Seqüência – Consultar Docente

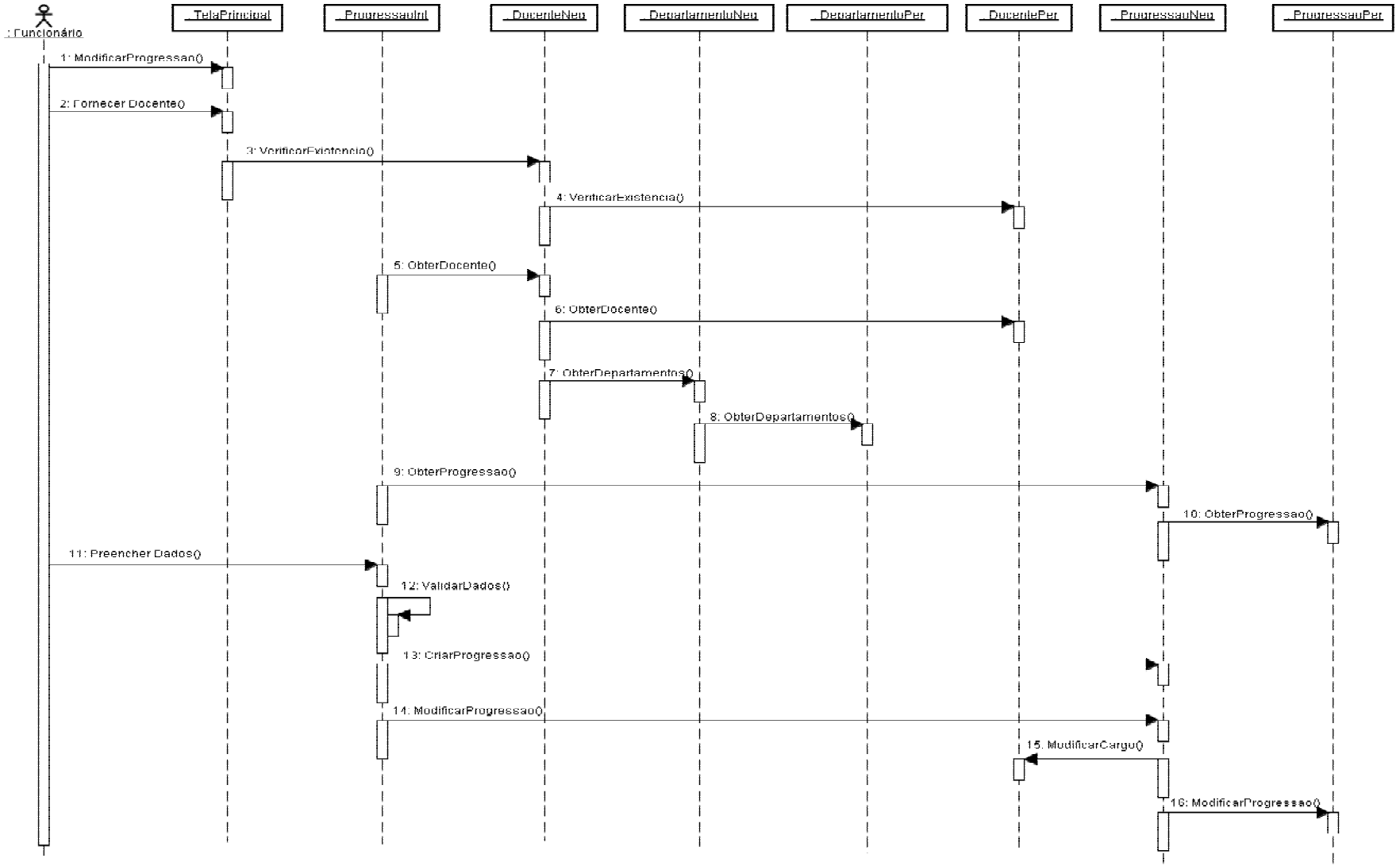


Figura 6.22 – Diagrama de Seqüência – Modificar Progressão de Docente

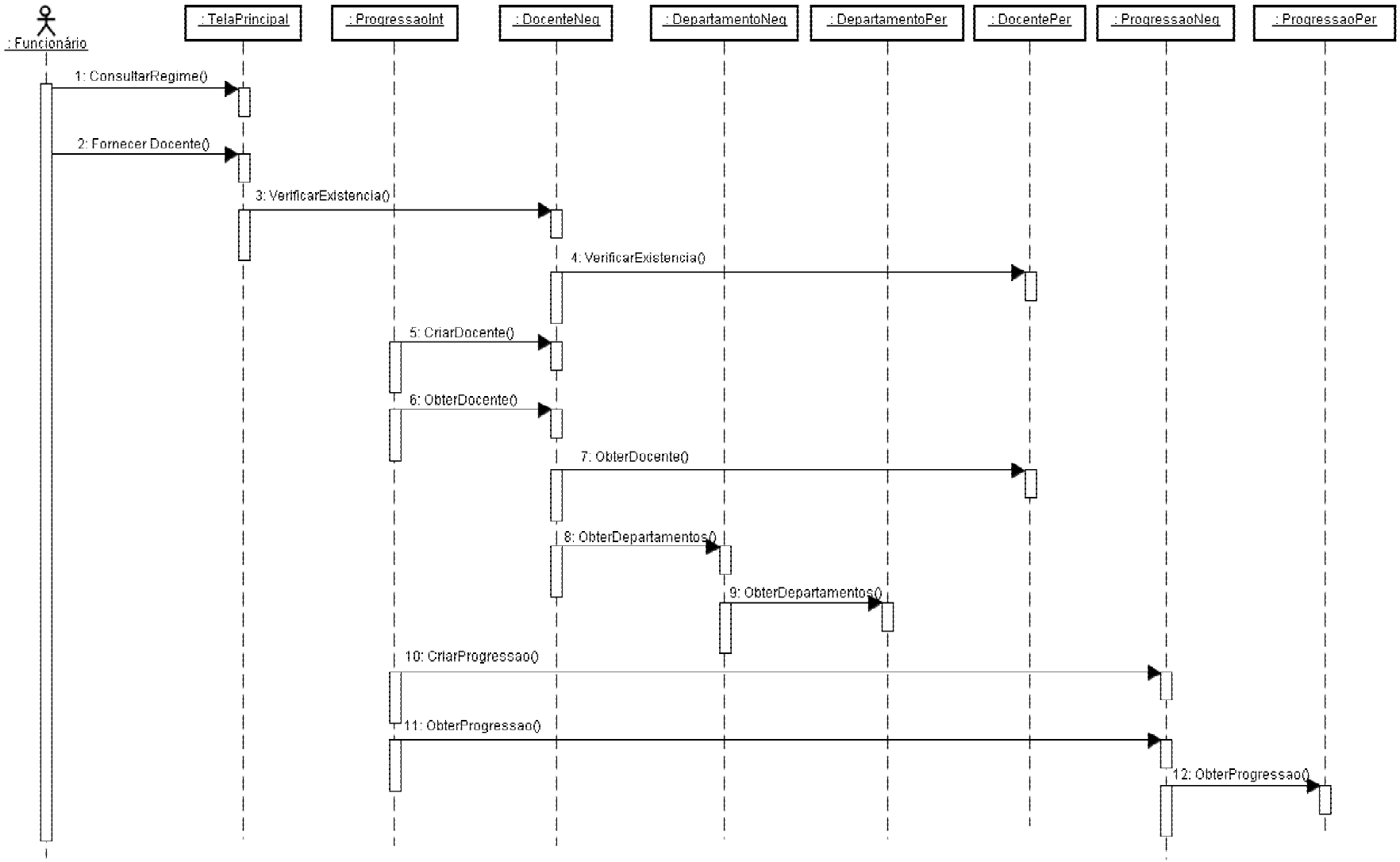


Figura 6.23 – Diagrama de Seqüência – Consultar Progressão de Docente

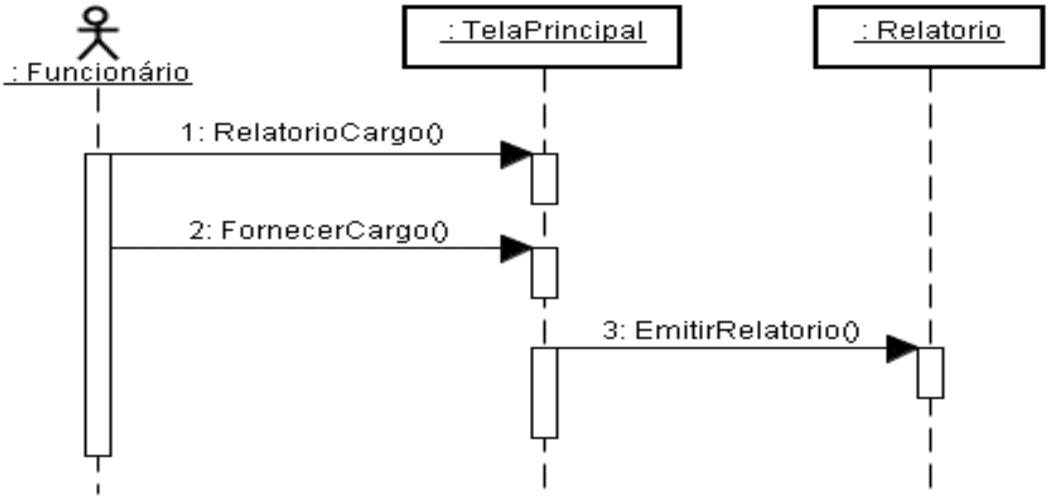


Figura 6.24 – Diagrama de Seqüência – Emitir Relatório por Cargo

6.3.4 Apresentação do Sistema

Nesta seção, é apresentado o produto de software descrevendo algumas situações de uso.

Mediante autenticação, a tela principal é exibida ao usuário que permite acesso a todos os menus do produto de software, sendo possível realizar as seguintes operações:

- Cadastro: incluir, alterar, excluir, consultar docente;
- Progressão: modificar, consultar progressão;
- Regime: modificar, consultar regime;
- Departamento: incluir, alterar, excluir departamento;
- Relatório: emissão de relatórios;
- Ajuda: informações sobre o sistema.

A Figura 6.25 ilustra a tela exibida ao selecionar a operação alterar departamento, dentro do contexto do caso de uso Manter Cadastro de Departamento. Após selecionar o departamento desejado, é possível modificar sua sigla e/ou descrição. Seguindo os passos do Diagrama de Seqüência correspondente, verifica-se que a interface comunica-se com a lógica de negócio que, por sua vez, se comunica com a camada de persistência, obtendo as informações de todos os departamentos presentes no banco de dados.

Através do menu Cadastro, é possível executar a operação Incluir Docente. A Figura 6.26 apresenta a tela de inclusão. Após preencher os dados e clicar em Incluir, é realizada a

validação dos dados, verificação da existência do docente e, por fim, os dados são incluídos no banco. Automaticamente, são criados seu regime e sua progressão.

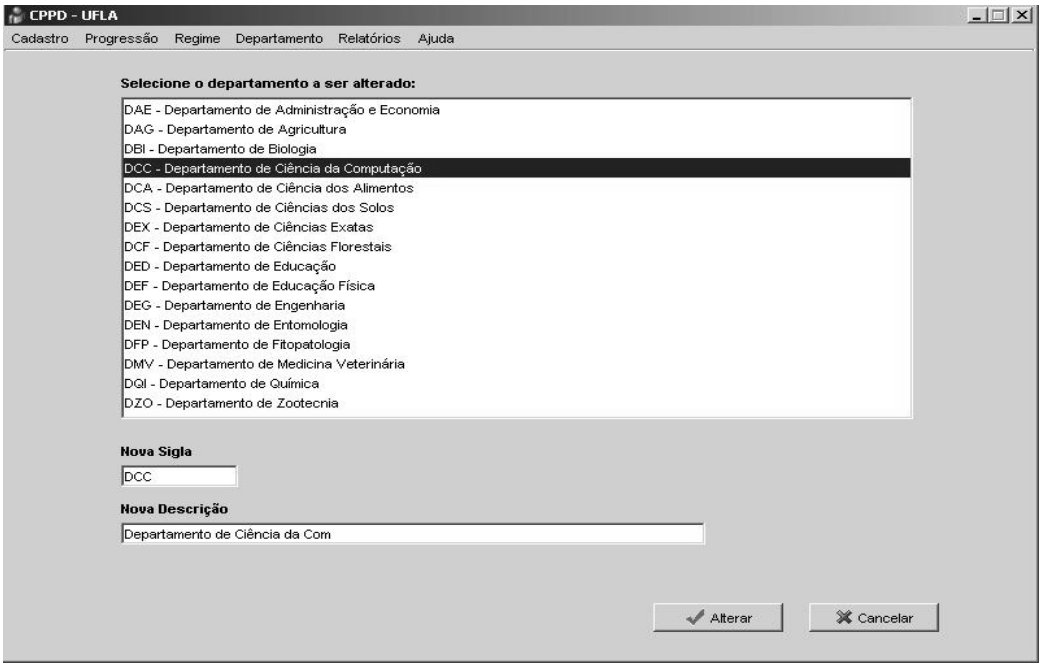


Figura 6.25 – Operação Alterar Departamento

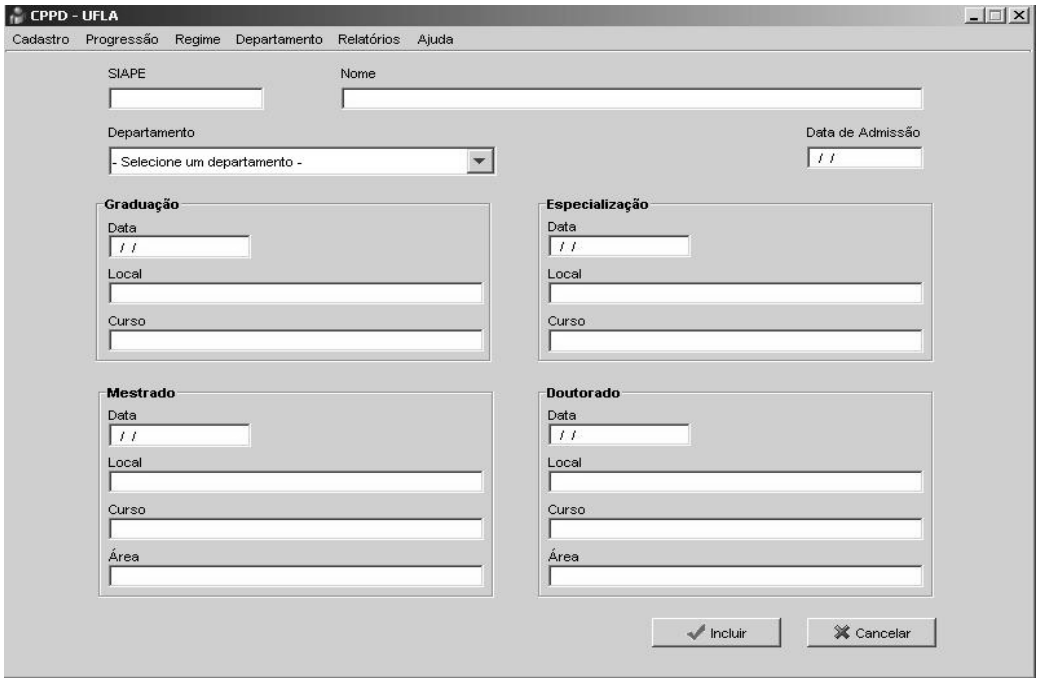


Figura 6.26 – Operação Incluir Docente

A progressão dos docentes é modificada através do menu Progressão. Após fornecer o número siape ou nome do docente, a tela de progressão é exibida. Assim como na modificação de regime de docentes, os dados são validados e, posteriormente, alterados no banco de dados. A Figura 6.27 representa a tela para modificar a progressão.

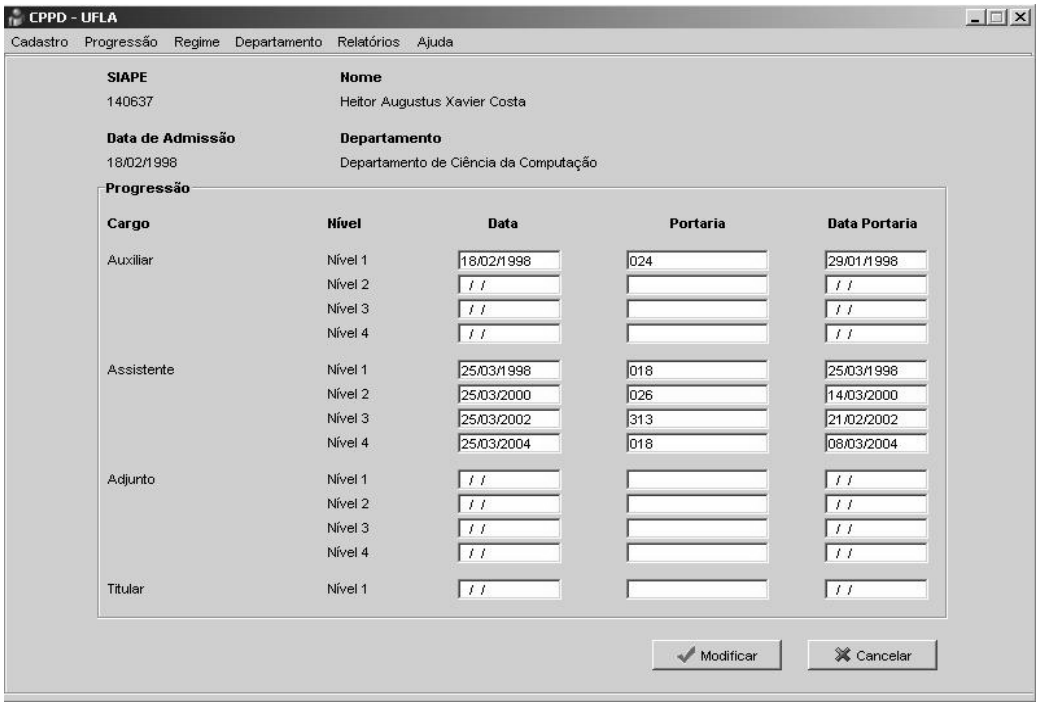


Figura 6.27 – Operação Modificar Progressão de Docente

A emissão de relatórios por cargo, por departamento, por titulação, por data de admissão e por progressão do mês é possível através do menu Relatórios. A Figura 6.28 ilustra a tela exibida ao usuário ao emitir um relatório por departamento. No caso, o departamento selecionado foi Ciência da Computação.

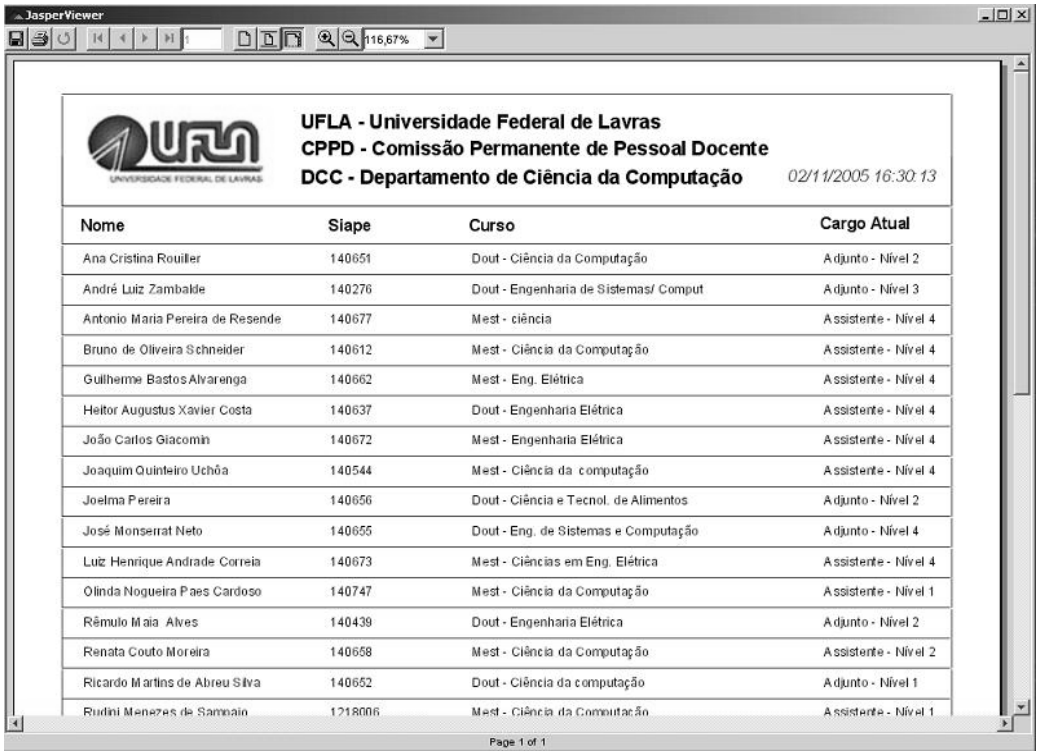


Figura 6.28 – Operação Emitir Relatório por Departamento

6.4 Produto de Software: CIS

A Comissão Interna de Supervisão, antiga Comissão Permanente de Pessoal Técnico Administrativo (CPPTA), da Universidade Federal de Lavras possui, dentre suas funções, o objetivo de manter um cadastro dos técnicos administrativos da universidade, bem como avaliar o desempenho de cada técnico administrativo para progressão funcional.

O produto de software CIS visa facilitar este cadastro e o controle dos demais processos dos técnicos administrativos.

6.4.1 Modelagem dos Dados

A Figura 6.29 apresenta o Modelo Entidade Relacionamento do produto de software CIS.

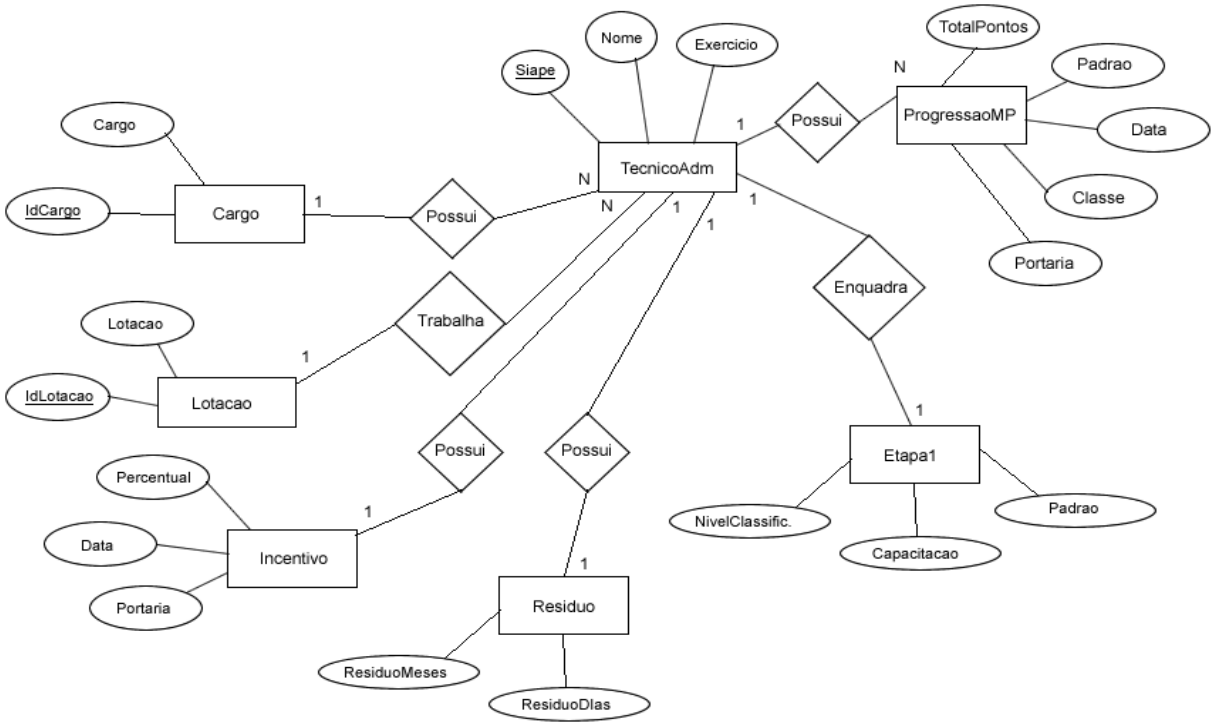


Figura 6.29 – Modelo Entidade-Relacionamento – CIS

6.4.2 Desenvolvimento

Nesta seção, é descrito como o produto de software CIS foi desenvolvido, descrevendo cada uma das etapas do processo de desenvolvimento pelo qual ele passou.

6.4.2.1 Análise de Requisitos

Durante esta fase foram criados o Diagrama de Casos de Uso e a Descrição dos Casos de Uso descritos nas seções subseqüentes.

6.4.2.1.1 Diagrama de Casos de Uso

A Figura 6.30 ilustra o Diagrama de Casos de Uso com o ator funcionário e os casos de uso do Produto de Software CIS: Manter Cadastro de Cargo, Manter Cadastro de Lotação, Manter Cadastro de Técnico Administrativo, Modificar Progressão por Mérito Profissional, Consultar Progressão por Mérito Profissional, Emitir Relatório por Cargo, Emitir Relatório por Lotação, Emitir Relatório por Exercício, Emitir Relatório por Progressão do Mês.

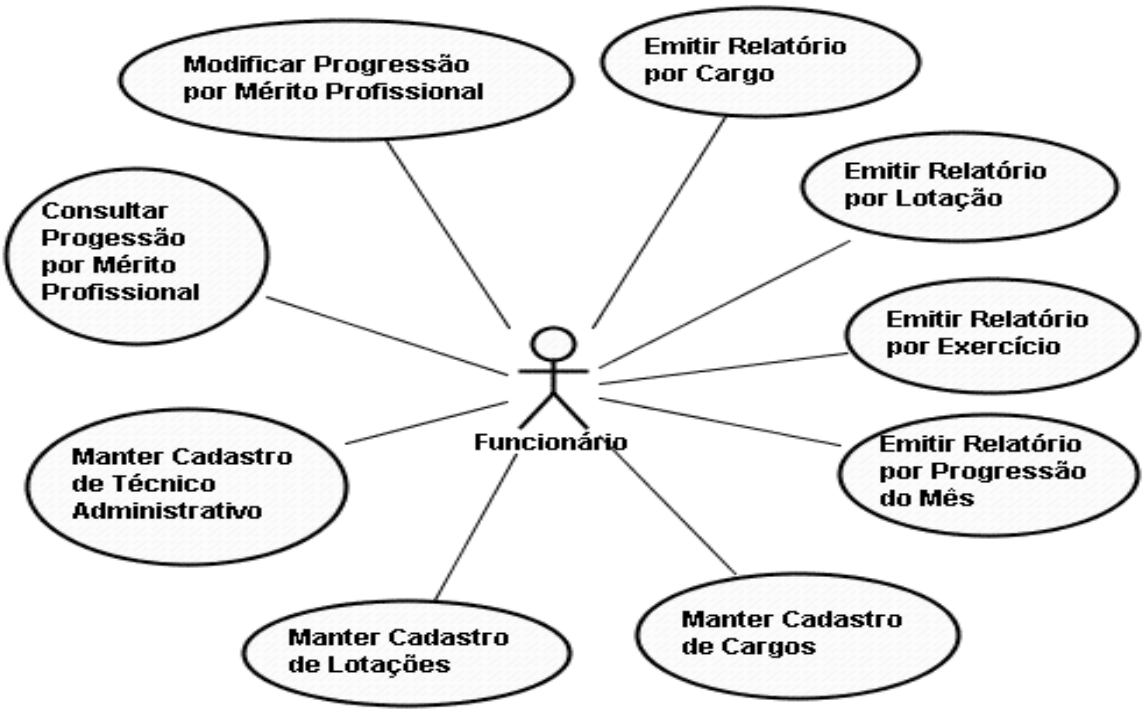


Figura 6.30 – Diagrama de Casos de Uso

Os casos de uso Manter Cadastro de Cargo e Manter Cadastro de Lotação englobam as operações Inserir, Alterar e Excluir. O caso de uso Manter Cadastro de Técnico Administrativo contém as operações anteriores, acrescentadas da operação Consultar.

Assim como no sistema CPPD, tais operações foram definidas pelo cliente durante as reuniões realizadas.

6.4.2.1.2 Descrição dos Casos de Uso

A Descrição dos Casos de Uso tem por objetivo registrar de forma textual, o comportamento de cada caso de uso, acrescentando detalhes que enriquecem e tornam mais claros e completos os casos de uso. Estas informações são:

- **Id**: Identificador numérico único;
- **Nome**: Nome único para descrever a função do caso de uso;
- **Descrição**: Descrição sucinta das atividades que devem ser consideradas;
- **Ator**: Nome do ator(es) que interage(m) com o produto de software;
- **Pré-Condições**: Condições em que o produto de software de estar para iniciar o caso de uso principal;
- **Execução Normal**: Seqüência de passos que devem ocorrer quando o produto de software atender a todas as pré-condições, com as mensagens necessárias ao(s) ator(es);
- **Pós-Condições**: Condições em que o produto de software deve estar após a execução do caso de uso;
- **Exceções**: seqüência de passos que devem ocorrer quando alguma pré-condição não for verificada, com as mensagens necessárias ao(s) ator(es).

A seguir, é apresentada a Descrição do Caso de Uso dos seguintes casos de uso:

- Manter Cadastro de Cargo (Tabela 6.30);
- Manter Cadastro de Lotação (Tabela 6.31);
- Manter Cadastro de Técnico Administrativo (Tabela 6.32);
- Modificar Progressão por Mérito Profissional (Tabela 6.33);
- Consultar Progressão por Mérito Profissional (Tabela 6.34);
- Emitir Relatório por Cargo (Tabela 6.35);
- Emitir Relatório por Lotação (Tabela 6.36);
- Emitir Relatório por Exercício (Tabela 6.37);
- Emitir Relatório por Progressão do Mês (Tabela 6.38).

Tabela 6.30 – Descrição do Caso de Uso Manter Cadastro de Cargo

Id Caso de Uso: 001	
Nome: Manter Cadastro de Cargo	
Descrição: Permite realizar as operações de Inclusão, Alteração e Exclusão de Cargos.	
Ator: Funcionário	
Incluir Cargo	
Pré-Condições	O cargo não deve existir no sistema.
Execução Normal	A inclusão do cargo é feita com a entrada de seus dados.
Pós-Condições	O sistema emite uma mensagem indicando que a inclusão foi efetuada com sucesso.
Exceções	Se o cargo existir no sistema, a inclusão não é realizada e o sistema emite uma mensagem indicando que a inclusão não foi efetuada.
Alterar Cargo	
Pré-Condições	O funcionário deve selecionar o cargo a ser alterado.
Execução Normal	O funcionário altera os dados do cargo previamente selecionado.
Pós-Condições	O sistema emite uma mensagem indicando que as alterações foram efetuadas com sucesso.
Exceções	Se o funcionário não selecionar algum cargo, é emitida uma mensagem indicando que algum cargo deve ser selecionado.
Excluir Cargo	
Pré-Condições	O funcionário deve selecionar o cargo a ser excluído.
Execução Normal	O funcionário seleciona o cargo a ser excluído.
Pós-Condições	O sistema emite uma mensagem indicando que o cargo foi excluído com sucesso.
Exceções	Se o funcionário não selecionar algum cargo, é emitida uma mensagem indicando que algum cargo deve ser selecionado.

Tabela 6.31 – Descrição do Caso de Uso Manter Cadastro de Lotação

Id Caso de Uso: 002	
Nome: Manter Cadastro de Lotação	
Descrição: Permite realizar as operações de Inclusão, Alteração e Exclusão de Lotações.	
Ator: Funcionário	
Incluir Lotação	
Pré-Condições	A lotação não deve existir no sistema.
Execução Normal	A inclusão da lotação é feita com a entrada de seus dados.
Pós-Condições	O sistema emite uma mensagem indicando que a inclusão foi efetuada com sucesso.
Exceções	Se a lotação existir no sistema, a inclusão não é realizada e o sistema emite uma mensagem indicando que a inclusão não foi efetuada.
Alterar Lotação	
Pré-Condições	O funcionário deve selecionar a lotação a ser alterada.
Execução Normal	O funcionário altera os dados da lotação previamente selecionada.
Pós-Condições	O sistema emite uma mensagem indicando que as alterações foram efetuadas com sucesso.

Exceções	Se o funcionário não selecionar alguma lotação, é emitida uma mensagem indicando que alguma lotação deve ser selecionada.
Excluir Lotação	
Pré-Condições	O funcionário deve selecionar a lotação a ser excluída.
Execução Normal	O funcionário seleciona a lotação a ser excluída.
Pós-Condições	O sistema emite uma mensagem indicando que a lotação foi excluída com sucesso.
Exceções	Se o funcionário não selecionar alguma lotação, é emitida uma mensagem indicando que alguma lotação deve ser selecionada.

Tabela 6.32 – Descrição do Caso de Uso Manter Cadastro de Técnico Administrativo

Id Caso de Uso: 003	
Nome: Manter Cadastro de Técnico Administrativo	
Descrição: Permite realizar as operações de Inclusão, Alteração, Exclusão e Consulta de Técnicos Administrativos.	
Ator: Funcionário	
Incluir Técnico Administrativo	
Pré-Condições	O técnico administrativo não deve existir no sistema.
Execução Normal	A inclusão do técnico administrativo é feita com a entrada de seus dados.
Pós-Condições	O sistema emite uma mensagem indicando que a inclusão foi efetuada com sucesso. Automaticamente, são criados sua progressão e seu regime.
Exceções	Se o técnico administrativo existir no sistema, a inclusão não é realizada e o sistema emite uma mensagem indicando que a inclusão não foi efetuada.
Alterar Técnico Administrativo	
Pré-Condições	O técnico administrativo deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do técnico administrativo, alterando os dados desejados posteriormente.
Pós-Condições	O sistema emite uma mensagem indicando que as alterações foram efetuadas com sucesso.
Exceções	Se o técnico administrativo não existir no sistema, é emitida uma mensagem indicando que o técnico administrativo não existe.
Excluir Técnico Administrativo	
Pré-Condições	O técnico administrativo deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do técnico administrativo. Uma mensagem de confirmação é exibida. Automaticamente, são excluídos sua progressão e seu regime.
Pós-Condições	O sistema emite uma mensagem indicando que a exclusão foi concluída com sucesso.
Exceções	Se o técnico administrativo não existir no sistema, é emitida uma mensagem indicando que o técnico administrativo não existe.
Consultar Técnico Administrativo	
Pré-Condições	O técnico administrativo deve existir no sistema.
Execução Normal	O funcionário realiza uma consulta sobre algum técnico administrativo informando seu número siape ou seu nome.

Pós-Condições	Os dados do técnico administrativo são exibidos.
Exceções	Se o técnico administrativo não existir no sistema, é emitida uma mensagem indicando que o técnico administrativo não existe.

Tabela 6.33 – Descrição do Caso de Uso Modificar Progressão por Mérito Profissional

Id Caso de Uso: 004	
Nome: Modificar Progressão por Mérito Profissional	
Descrição: Permite modificar a progressão por mérito profissional dos técnicos administrativos cadastrados.	
Ator: Funcionário	
Pré-Condições	O técnico administrativo deve existir no sistema.
Execução Normal	Após informar o número siape ou nome do técnico administrativo, é possível modificar os dados de sua progressão.
Pós-Condições	É emitida uma mensagem indicando que seus dados foram modificados com sucesso.
Exceções	Se o técnico administrativo não existir no sistema, é emitida uma mensagem indicando que o técnico administrativo não existe.

Tabela 6.34 – Descrição do Caso de Uso Consultar Progressão por Mérito Profissional

Id Caso de Uso: 005	
Nome: Consultar Progressão por Mérito Profissional	
Descrição: Permite consultar a progressão por mérito profissional dos técnicos administrativos cadastrados.	
Ator: Funcionário	
Pré-Condições	O técnico administrativo deve existir no sistema.
Execução Normal	O funcionário informa o número siape ou o nome do técnico administrativo.
Pós-Condições	Os dados da progressão do técnico administrativo são exibidos.
Exceções	Se o técnico administrativo não existir no sistema, é emitida uma mensagem indicando que o técnico administrativo não existe.

Tabela 6.35 – Descrição do Caso de Uso Emitir Relatório por Cargo

Id Caso de Uso: 006	
Nome: Emitir Relatório por Cargo	
Descrição: Emite um relatório contendo informações dos técnicos administrativos que possuem o mesmo cargo. Estas informações são nome, SIAPE, lotação e exercício.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar algum cargo, sendo que há algum pré-selecionado.
Execução Normal	O funcionário seleciona o cargo desejado e a ordem de exibição dos técnicos administrativos, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório.
Exceções	Se não existirem técnicos administrativos referentes ao cargo selecionado, é emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.36 – Descrição do Caso de Uso Emitir Relatório por Lotação

Id Caso de Uso: 007	
Nome: Emitir Relatório por Lotação	
Descrição: Emite um relatório contendo informações dos técnicos administrativos que trabalham na mesma lotação. Estas informações são nome, SIAPE, cargo e exercício.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar alguma lotação, sendo que há alguma pré-selecionada.
Execução Normal	O funcionário informa a lotação e a ordem de exibição dos técnicos administrativos, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório.
Exceções	Se não existirem técnicos administrativos referentes à lotação selecionada, é emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.37 – Descrição do Caso de Uso Emitir Relatório por Exercício

Id Caso de Uso: 008	
Nome: Emitir Relatório por Exercício	
Descrição: Emite um relatório contendo informações dos técnicos administrativos, que foram admitidos em determinada data (mês e ano). Estas informações são nome, SIAPE, lotação e exercício.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar o mês e informar o ano.
Execução Normal	O funcionário informa a data (mês e ano) desejada.
Pós-Condições	O sistema emite o relatório ordenado por nome de técnico administrativo.
Exceções	O relatório não é emitido se não houverem técnicos administrativos admitidos na data previamente informada, sendo emitida uma mensagem indicando que o relatório é vazio.

Tabela 6.38 – Descrição do Caso de Uso Emitir Relatório por Progressão do Mês

Id Caso de Uso: 009	
Nome: Emitir Relatório por Progressão do Mês	
Descrição: Emite um relatório contendo informações dos técnicos administrativos, que terão progressão em determinada data (mês e ano). Estas informações são nome, SIAPE, lotação, cargo, nível, capacitação, próximo padrão e próxima progressão.	
Ator: Funcionário	
Pré-Condições	O funcionário deve selecionar o mês e informar o ano.
Execução Normal	O funcionário informa a data (mês e ano) desejada e a ordem de exibição dos técnicos administrativos, podendo ser SIAPE ou Nome.
Pós-Condições	O sistema emite o relatório ordenado por nome de técnico administrativo.
Exceções	O relatório não é emitido se não houver técnicos administrativos que terão progressão na data previamente informada, sendo emitida uma mensagem indicando que o relatório é vazio.

6.4.2.2 Análise

O Modelo de Análise apresentado nesta seção é composto pelo Diagrama de Classes, o Dicionário de Atributos e Métodos e os seguintes Diagramas de Seqüência: Incluir Cargo, Alterar Cargo, Excluir Cargo, Incluir Lotação, Alterar Lotação, Excluir Lotação, Incluir Técnico Administrativo, Alterar Técnico Administrativo, Excluir Técnico Administrativo, Consultar Técnico Administrativo, Modificar Progressão por Mérito Profissional de Técnico Administrativo, Consultar Progressão por Mérito Profissional de Técnico Administrativo, Emitir Relatório por Cargo, Emitir Relatório por Lotação, Emitir Relatório por Exercício e Emitir Relatório por Progressão do Mês.

6.4.2.2.1 Diagrama de Classes

O Diagrama de Classes representa uma visão estática do sistema, porque a estrutura e o comportamento descritos são sempre válidos em qualquer ponto do ciclo de vida do produto de software. A Figura 6.31 ilustra o Diagrama de Classes do Produto de Software CPPD, para sua representação foram utilizados os elementos de modelagem sugeridos pela UML (BOOCH *et al*, 1999).

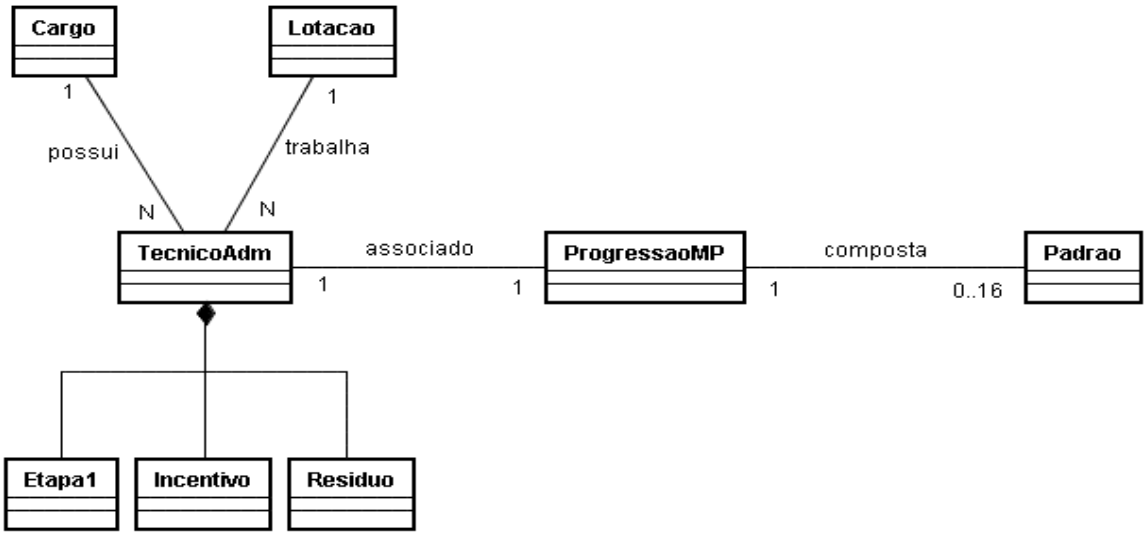


Figura 6.31 – Diagrama de Classes – Modelo de Análise

6.4.2.2.2 Dicionário de Atributos e Métodos

Nesta seção, é apresentado o Dicionário de Atributos e Métodos, que fornece uma descrição detalhada das classes. Essa descrição consiste em organizar os atributos e os métodos das classes, apresentando suas características. Além disso, contribui para não

sobrecarregar visualmente o Diagrama de Classes. A seguir é apresentado o Dicionário de Atributos e Métodos das seguintes classes:

- Etapa1 (Tabela 6.39);
- Incentivo (Tabela 6.40);
- Residuo (Tabela 6.41);
- Cargo (Tabela 6.42);
- TecnicoAdm (Tabela 6.43);
- Padrao (Tabela 6.44);
- ProgressaoMP (Tabela 6.45).

As classes Cargo e Lotação são semelhantes.

Tabela 6.39 – Dicionário de Atributos e Métodos da Classe Etapa1

Classe	**Etapa1**				
Descrição	Identifica e manipula informações sobre a 1ª etapa da progressão dos técnicos admnistrativos				
Atributos					
Identificador	nivelClassificacao				
Descrição	Nível de classificação relacionado à primeira etapa				
Valores Possíveis	Qualquer Letra ou Dígito				
Tamanho	1	**Formato**	Uma posição	**TAD**	Caractere
Identificador	capacitacao				
Descrição	Capacitação relacionada à primeira etapa				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	3	**Formato**	Letras	**TAD**	String
Identificador	padrao				
Descrição	Padrão relacionado à primeira etapa				
Valores Possíveis	1 a 2147483647				
Tamanho	2	**Formato**	Números	**TAD**	Inteiro
Métodos					
Identificador	criarEtapa1				
Descrição Funcional	Cria um objeto 1aEtapa				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	setNivelClassificacao				
Descrição Funcional	Seta o nível de classificação relacionado à primeira etapa				
Parâmetros de Entrada	nivelClassificacao		**TAD**	Caractere	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getCurso				
Descrição Funcional	Obtém o nível de classificação relacionado à primeira etapa				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	nivelClassificacao		**TAD**	Caractere	

Identificador	setCapacitacao		
Descrição Funcional	Seta a capacitação relacionada à primeira etapa		
Parâmetros de Entrada	capacitacao	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getCapacitacao		
Descrição Funcional	Obtém a capacitação relacionada à primeira etapa		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	capacitacao	**TAD**	String
Identificador	setPadrao		
Descrição Funcional	Seta o padrão relacionado à primeira etapa do padrão		
Parâmetros de Entrada	padrao	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getPadrao		
Descrição Funcional	Obtém o padrão relacionado à primeira etapa do padrão		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	padrao	**TAD**	Inteiro

Tabela 6.40 – Dicionário de Atributos e Métodos da Classe Incentivo

Classe	**Incentivo**				
Descrição	Identifica e manipula informações sobre o incentivo dos técnicos administrativos.				
Atributos					
Identificador	incentivo				
Descrição	Indica se o técnico administrativo possui algum incentivo				
Valores Possíveis	Verdadeiro ou Falso				
Tamanho	-	**Formato**	Verdadeiro ou Falso	**TAD**	Booleano
Identificador	percentual				
Descrição	Percentual de incentivo do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Identificador	portaria				
Descrição	Portaria do incentivo do técnico administrativo				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	10	**Formato**	Letras e Números	**TAD**	String
Identificador	data				
Descrição	Data do incentivo do técnico administrativo				
Valores Possíveis	Datas				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	Data
Métodos					

Identificador	criarIncentivo		
Descrição Funcional	Cria um objeto Incentivo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	setIncentivo		
Descrição Funcional	Seta se o técnico administrativo possui incentivo ou não		
Parâmetros de Entrada	incentivo	**TAD**	Booleano
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getIncentivo		
Descrição Funcional	Obtém se o técnico administrativo possui incentivo ou não		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	incentivo	**TAD**	Booleano
Identificador	setPercentual		
Descrição Funcional	Seta o percentual de incentivo do técnico administrativo		
Parâmetros de Entrada	percentual	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getPercentual		
Descrição Funcional	Obtém o percentual de incentivo do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	percentual	**TAD**	Inteiro
Identificador	setPortaria		
Descrição Funcional	Seta a portaria do incentivo		
Parâmetros de Entrada	portaria	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getPortaria		
Descrição Funcional	Obtém a portaria do incentivo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	portaria	**TAD**	String
Identificador	setData		
Descrição Funcional	Seta a data do incentivo		
Parâmetros de Entrada	data	**TAD**	Date
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getData		
Descrição Funcional	Obtém a data do incentivo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	data	**TAD**	Data

Tabela 6.41 – Dicionário de Atributos e Métodos da Classe Residuo

Classe	**Resíduo**				
Descrição	Identifica e manipula informações sobre os resíduos dos técnicos administrativos.				
Atributos					
Identificador	residuoMeses				
Descrição	Meses de resíduo do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Identificador	residuoDias				
Descrição	Dias de resíduo do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Métodos					
Identificador	criarResiduo				
Descrição Funcional	Cria um objeto Residuo				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	setResiduoMeses				
Descrição Funcional	Seta os meses do resíduo do técnico administrativo				
Parâmetros de Entrada	residuoMeses		**TAD**	Inteiro	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getResiduoMeses				
Descrição Funcional	Obtém os meses do resíduo do técnico administrativo				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	residuoMeses		**TAD**	Inteiro	
Identificador	setResiduoDias				
Descrição Funcional	Seta os dias do resíduo do técnico administrativo				
Parâmetros de Entrada	residuoDias		**TAD**	Inteiro	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getResiduoDias				
Descrição Funcional	Obtém os dias do resíduo do técnico administrativo				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	residuoDias		**TAD**	Inteiro	

Tabela 6.42 – Dicionário de Atributos e Métodos da Classe Cargo

Classe	**Cargo**
Descrição	Identifica e manipula informações sobre o cargo dos técnicos administrativos
Atributos	

Identificador	cargo				
Descrição	Cargo do técnico administrativo				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	60	**Formato**	Letras	**TAD**	String
Métodos					
Identificador	criarCargo				
Descrição Funcional	Cria um objeto Cargo				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	setCargo				
Descrição Funcional	Seta o cargo do técnico administrativo				
Parâmetros de Entrada	cargo	**TAD**	String		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	getCargo				
Descrição Funcional	Obtém o cargo do técnico administrativo				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		
Parâmetros de Saída	cargo	**TAD**	String		

Tabela 6.43 – Dicionário de Atributos e Métodos da Classe TecnicoAdm

Classe	**TecnicoAdm**				
Descrição	Identifica e manipula informações sobre os técnicos administrativos.				
Atributos					
Identificador	siape				
Descrição	Número siape do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	String
Identificador	nome				
Descrição	Nome do técnico administrativo				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	50	**Formato**	Letras	**TAD**	String
Identificador	exercicio				
Descrição	Data de ingresso do técnico administrativo na universidade				
Valores Possíveis	Datas				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	Data
Identificador	residuo				
Descrição	Resíduo do técnico administrativo				
TAD	Objeto Residuo				
Identificador	etapa1				
Descrição	Dados da 1ª etapa do técnico administrativo				
TAD	Objeto Etapa1				

Identificador	incentivo		
Descrição	Dados do incentivo do técnico administrativo		
TAD	Objeto Incentivo		
Métodos			
Identificador	criarTecnicoAdm		
Descrição Funcional	Cria um objeto TecnicoAdm		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	setSiape		
Descrição Funcional	Atribui o número siape do técnico administrativo		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getSiape		
Descrição Funcional	Obtém o número siape do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	siape	**TAD**	Inteiro
Identificador	setNome		
Descrição Funcional	Seta o nome do técnico administrativo		
Parâmetros de Entrada	nome	**TAD**	String
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getNome		
Descrição Funcional	Obtém o nome do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nome	**TAD**	String
Identificador	setExercicio		
Descrição Funcional	Seta o exercício do técnico administrativo		
Parâmetros de Entrada	exercicio	**TAD**	Data
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getExercicio		
Descrição Funcional	Obtém o exercício do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	exercicio	**TAD**	Data
Identificador	setResiduo		
Descrição Funcional	Seta o resíduo do técnico administrativo		
Parâmetros de Entrada	residuo	**TAD**	Objeto Residuo
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getResiduo		

Descrição Funcional	Obtém o resíduo do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	residuo	**TAD**	Objeto Residuo
Identificador	setEtapa1		
Descrição Funcional	Seta a 1ª etapa do técnico administrativo		
Parâmetros de Entrada	etapa1	**TAD**	Objeto Etapa1
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getEtapa1		
Descrição Funcional	Obtém a 1ª etapa do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	etapa1	**TAD**	Objeto Etapa1
Identificador	setIncentivo		
Descrição Funcional	Seta o incentivo do técnico administrativo		
Parâmetros de Entrada	incentivo	**TAD**	Objeto Incentivo
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getIncentivo		
Descrição Funcional	Obtém o incentivo do técnico administrativo		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	incentivo	**TAD**	Objeto Incentivo

Tabela 6.44 – Dicionário de Atributos e Métodos da Classe Padrao

Classe	**Padrao**				
Descrição	Identifica e manipula o padrão de progressão dos técnicos administrativos				
Atributos					
Identificador	data				
Descrição	Data do padrão				
Valores Possíveis	Datas				
Tamanho	10	**Formato**	DD/MM/AAAA	**TAD**	Data
Identificador	portaria				
Descrição	Portaria do padrão				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	20	**Formato**	Letras e Números	**TAD**	String
Identificador	classe				
Descrição	Classe do padrão				
Valores Possíveis	Qualquer seqüência de caracteres				
Tamanho	20	**Formato**	Letras	**TAD**	String
Identificador	totalPontos				
Descrição	Total de pontos atingido pelo técnico administrativo relativo ao padrão				
Valores Possíveis	Qualquer seqüência de caracteres				

Tamanho	10	**Formato**	Números	**TAD**	String
Métodos					
Identificador	criarPadrao				
Descrição Funcional	Cria um objeto Padrao				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	setData				
Descrição Funcional	Seta a data do padrão				
Parâmetros de Entrada	data	**TAD**	Data		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	getData				
Descrição Funcional	Obtém a data do padrão				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		
Parâmetros de Saída	data	**TAD**	Data		
Identificador	setPortaria				
Descrição Funcional	Seta a portaria do padrão				
Parâmetros de Entrada	portaria	**TAD**	String		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	getPortaria				
Descrição Funcional	Obtém a portaria do padrão				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		
Parâmetros de Saída	portaria	**TAD**	String		
Identificador	setClasse				
Descrição Funcional	Seta a classe do padrão				
Parâmetros de Entrada	classe	**TAD**	String		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	getClasse				
Descrição Funcional	Obtém a classe do padrão				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		
Parâmetros de Saída	classe	**TAD**	String		
Identificador	setTotalPontos				
Descrição Funcional	Seta o total de pontos atingido pelo técnico administrativo relativo ao padrão				
Parâmetros de Entrada	totalPontos	**TAD**	String		
Parâmetros de Saída	nenhum	**TAD**	nenhum		
Identificador	getTotalPontos				
Descrição Funcional	Obtém o total de pontos atingido pelo técnico administrativo relativo ao padrão				
Parâmetros de Entrada	nenhum	**TAD**	nenhum		

Parâmetros de Saída	totalPontos	**TAD**	String

Tabela 6.45 – Dicionário de Atributos e Métodos da Classe ProgressaoMP

Classe	**ProgressaoMP**		
Descrição	Identifica e manipula informações sobre a progressão por mérito profissional dos técnicos administrativos		
Atributos			
Identificador	padroes		
Descrição	Conjunto de padrões do técnico administrativo		
TAD	Vector de Objetos Padrao		
Métodos			
Identificador	criarProgressaoMP		
Descrição Funcional	Cria um objeto ProgressaoMP		
Parâmetros de Entrada	nenhum	**TAD**	nenhum
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	addPadrao		
Descrição Funcional	Adiciona um padrão ao conjunto de padrões do técnicos administrativos		
Parâmetros de Entrada	padrao	**TAD**	Objeto Padrao
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	getPadrao		
Descrição Funcional	Obtém um padrão ao conjunto de padrões do técnicos administrativos		
Parâmetros de Entrada	numPadrao	**TAD**	Inteiro
Parâmetros de Saída	padrao	**TAD**	Objeto Padrao

6.4.2.2.3 Diagramas de Seqüência

Um Diagrama de Seqüência descreve a maneira como os grupos de objetos colaboram em algum comportamento ao longo do tempo. Ele registra o comportamento de um único caso de uso, exibindo os objetos e as mensagens trocadas entre eles. A seguir, são apresentados os seguintes Diagramas de Seqüência:

- Incluir Cargo (Figura 6.32);
- Alterar Cargo (Figura 6.33);
- Excluir Cargo (Figura 6.34);
- Incluir Técnico Administrativo (Figura 6.35);
- Alterar Técnico Administrativo (Figura 6.36);
- Excluir Técnico Administrativo (Figura 6.37);
- Consultar Técnico Administrativo (Figura 6.38);

- Modificar Progressão por Mérito Profissional (Figura 6.39);
- Consultar Progressão por Mérito Profissional (Figura 6.40);
- Emitir Relatório por Exercício (Figura 6.41).

Os Diagramas de Seqüência para os casos de uso Manter Cadastro de Cargo e Lotação são semelhantes. Os Diagramas de Seqüência para os casos de uso de emissão de relatórios são semelhantes.

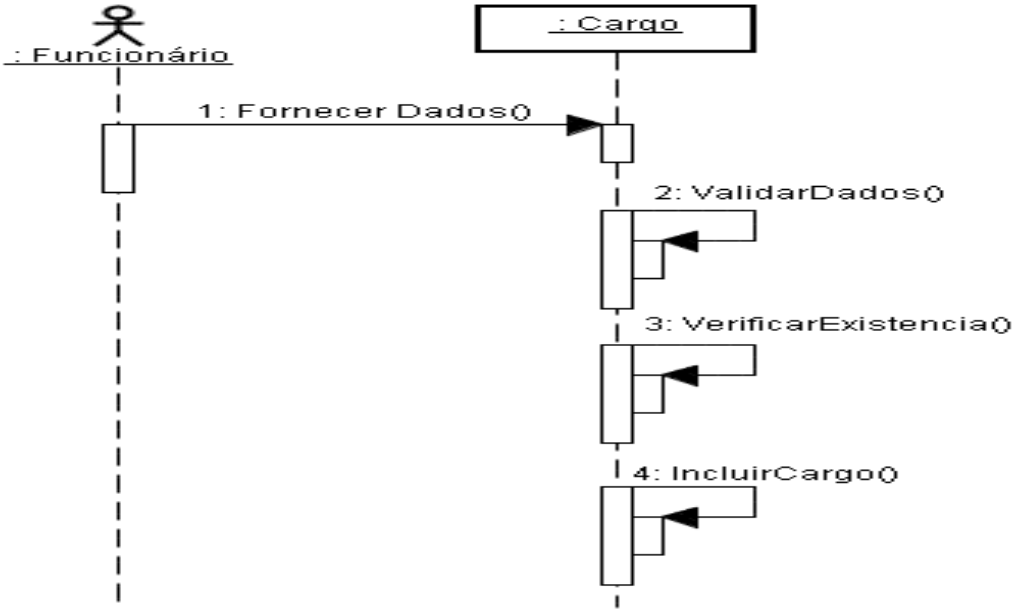


Figura 6.32 – Diagrama de Seqüência – Incluir Cargo

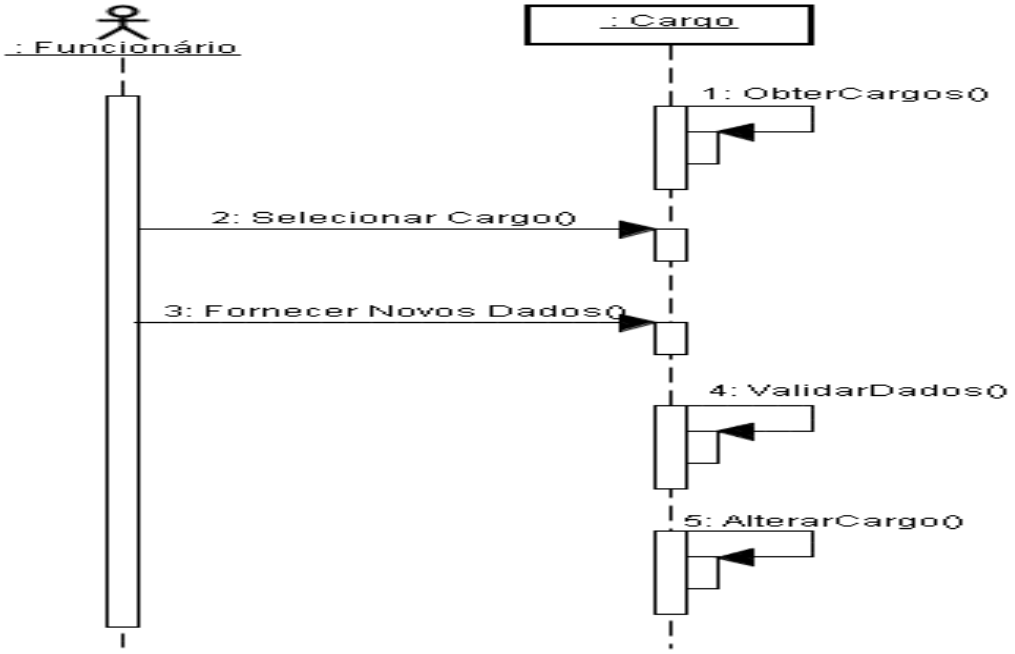


Figura 6.33 – Diagrama de Seqüência – Alterar Cargo

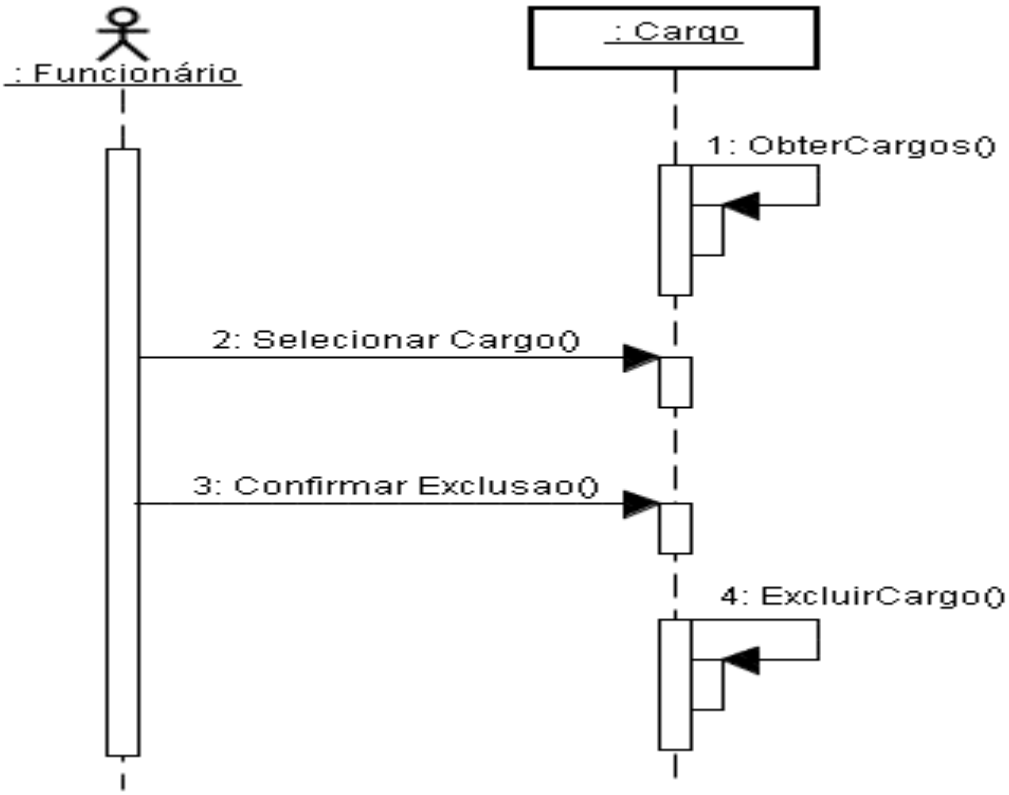


Figura 6.34 – Diagrama de Seqüência – Excluir Cargo

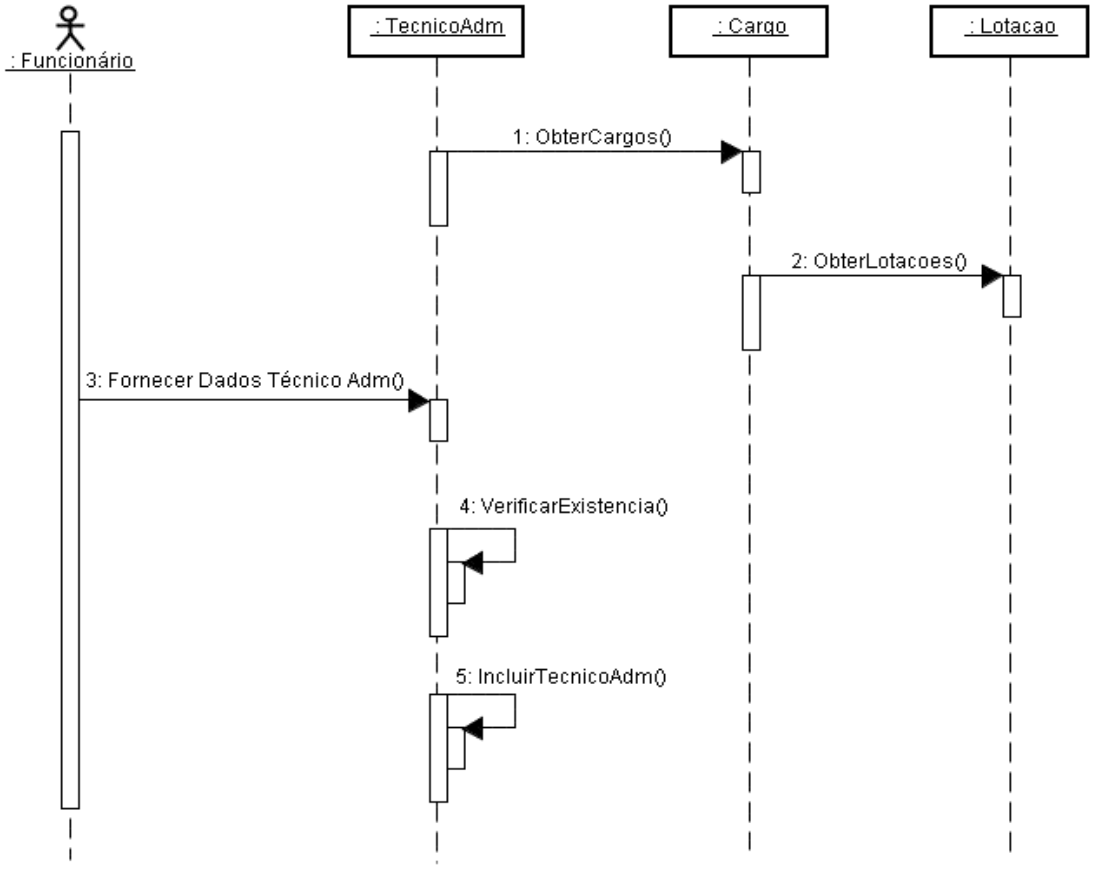


Figura 6.35 – Diagrama de Seqüência – Incluir Técnico Administrativo

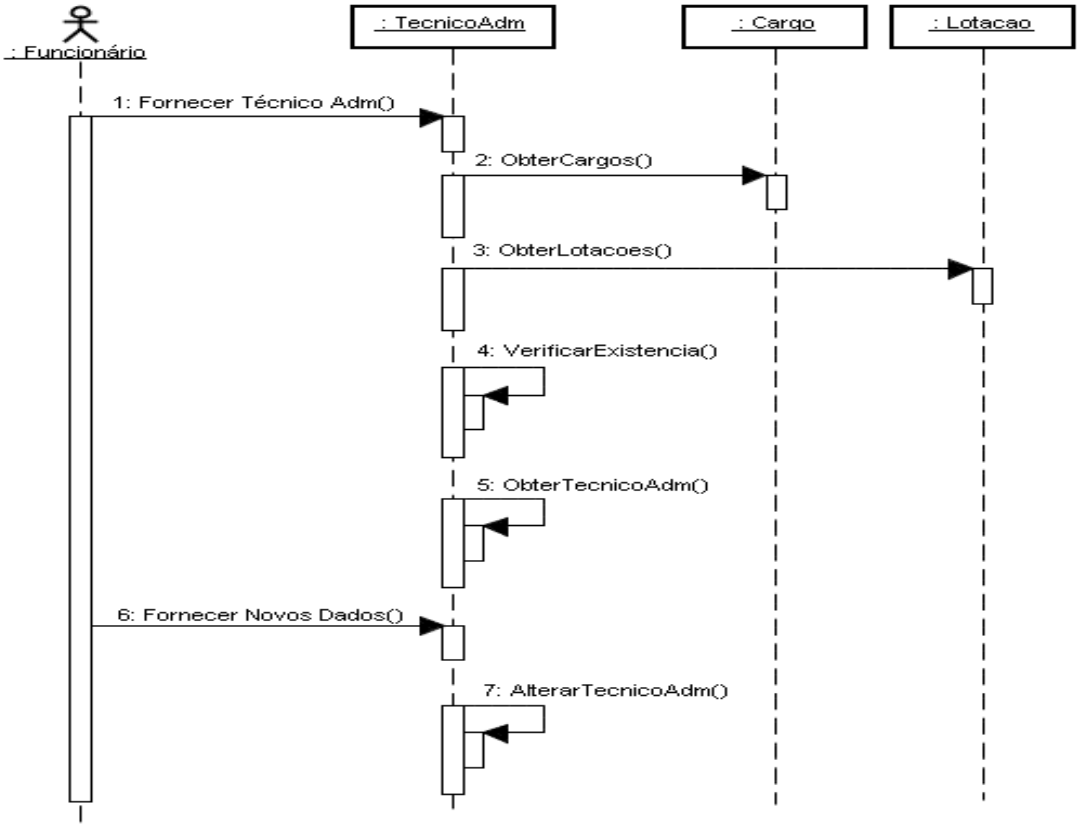


Figura 6.36 – Diagrama de Seqüência – Alterar Técnico Administrativo

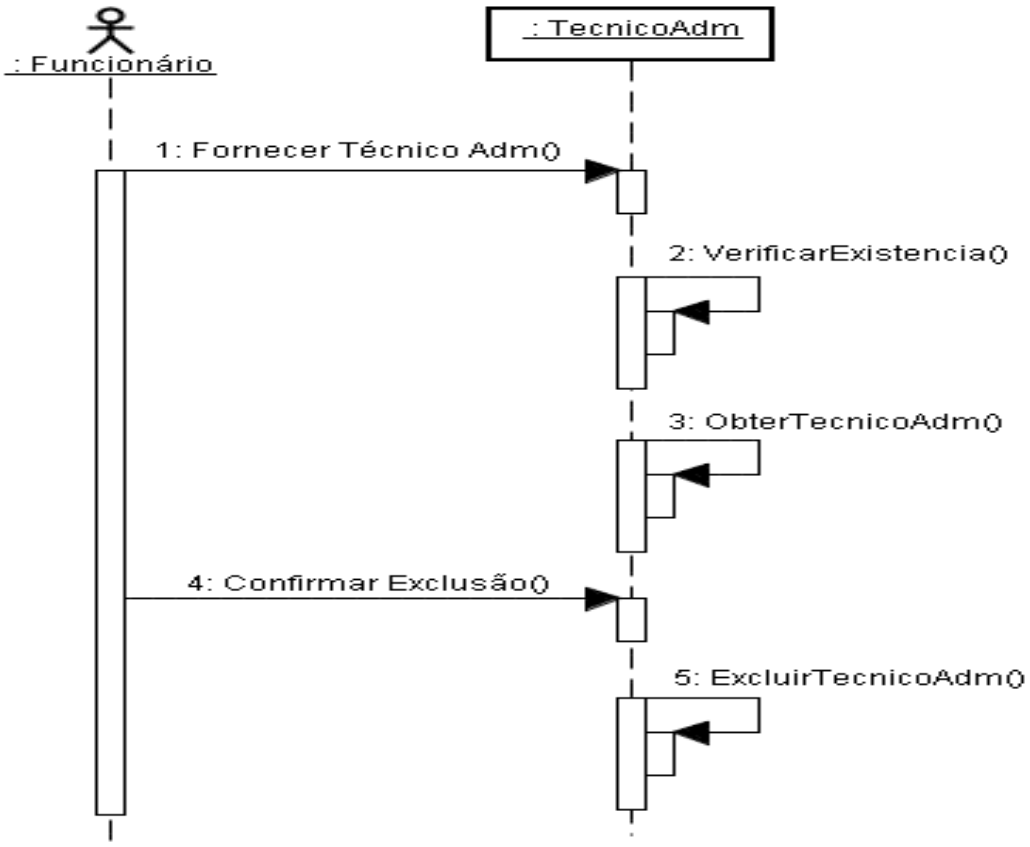


Figura 6.37 – Diagrama de Seqüência – Excluir Técnico Administrativo

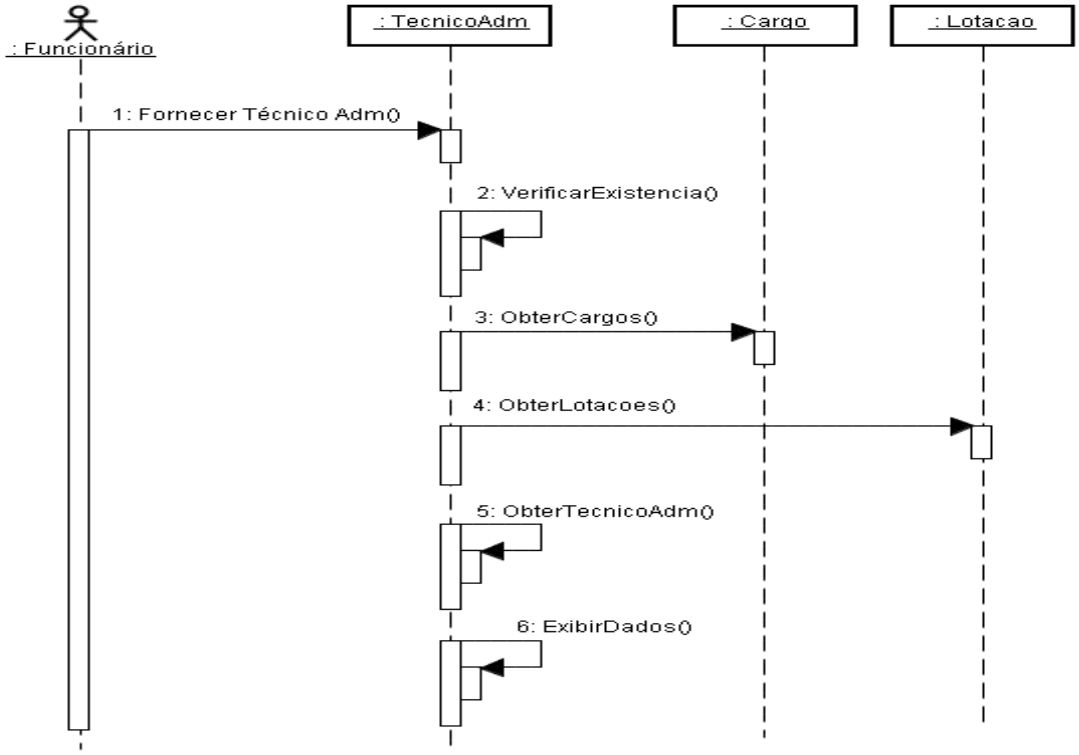


Figura 6.38 – Diagrama de Seqüência – Consultar Técnico Administrativo

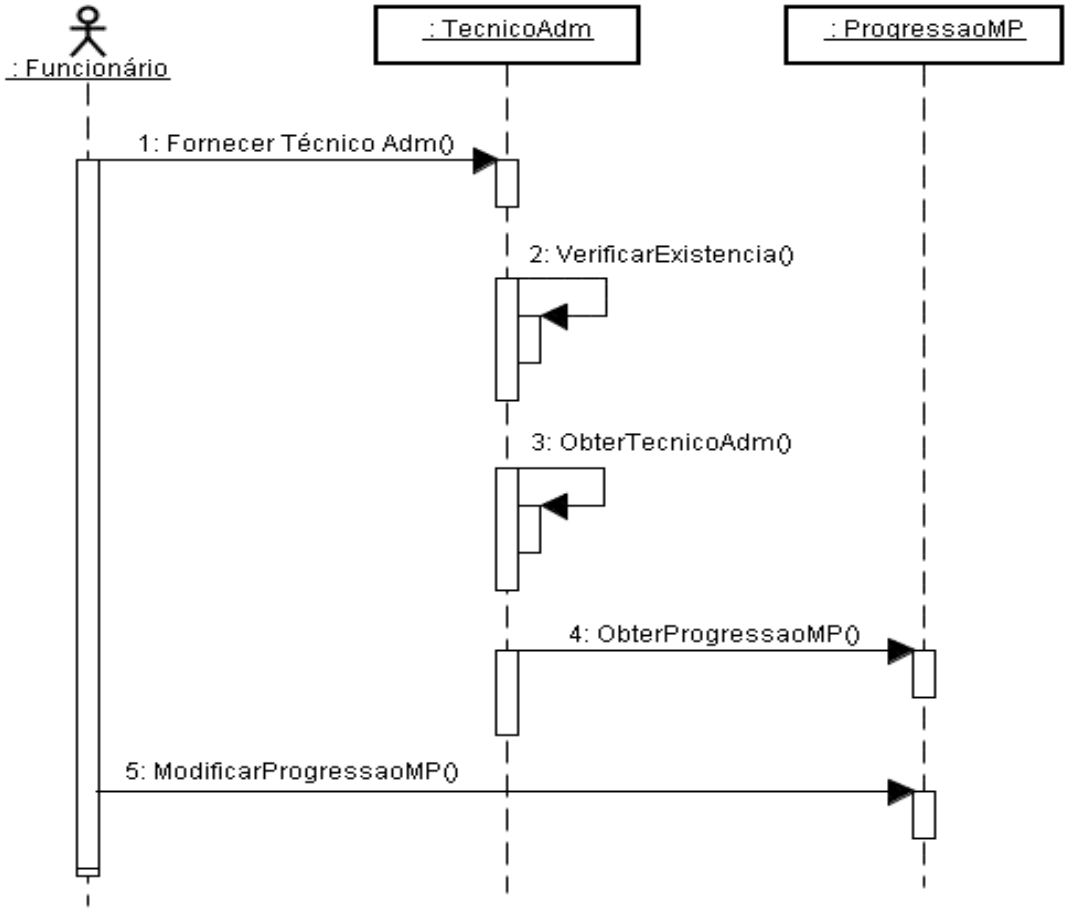


Figura 6.39 – Diagrama de Seqüência – Modificar Progressão por Mérito Profissional

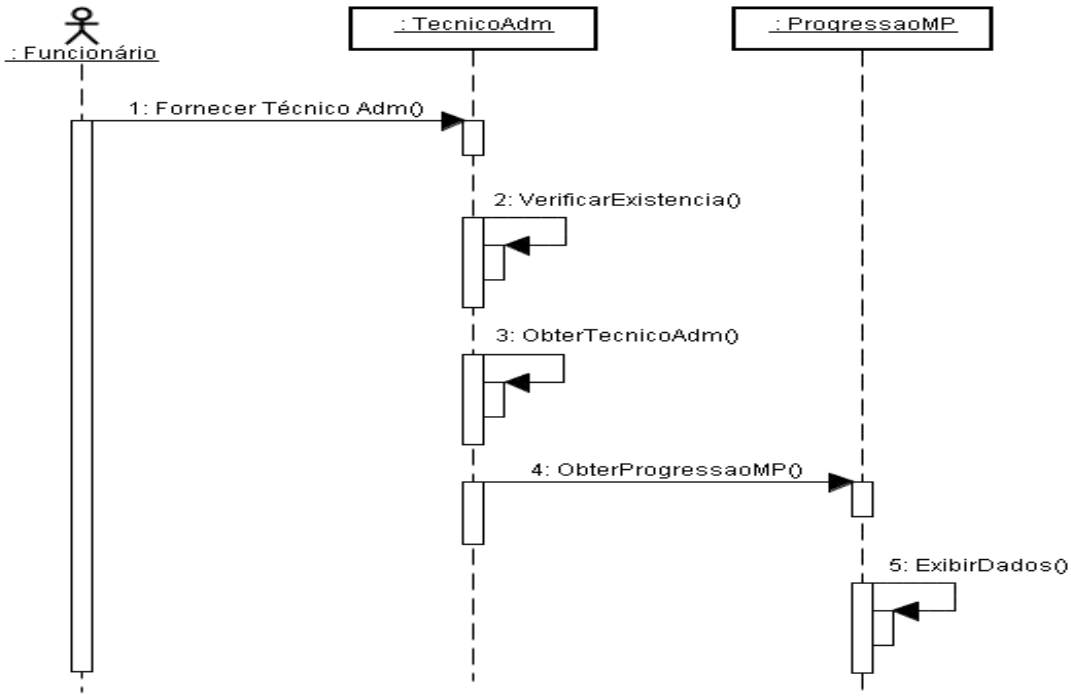


Figura 6.40 – Diagrama de Seqüência – Consultar Progressão por Mérito Profissional

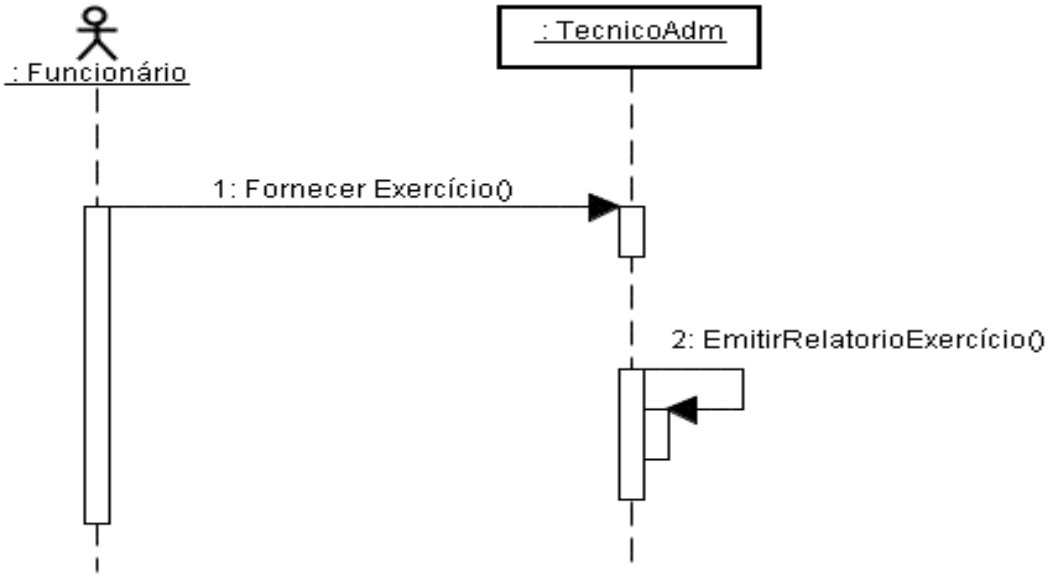


Figura 6.41 – Diagrama de Seqüência – Emitir Relatório por Exercício

6.4.2.3 Projeto

O Modelo de Projeto no contexto do trabalho é composto pelos mesmos diagramas do Modelo de Análise, porém engloba características de implementação (por exemplo, a interface do usuário e de periféricos, gerenciamento de banco de dados e comunicação com outros sistemas).

6.4.2.3.1 Diagrama de Classes

A Figura 6.42 ilustra o Diagrama de Classes do produto de software CIS.

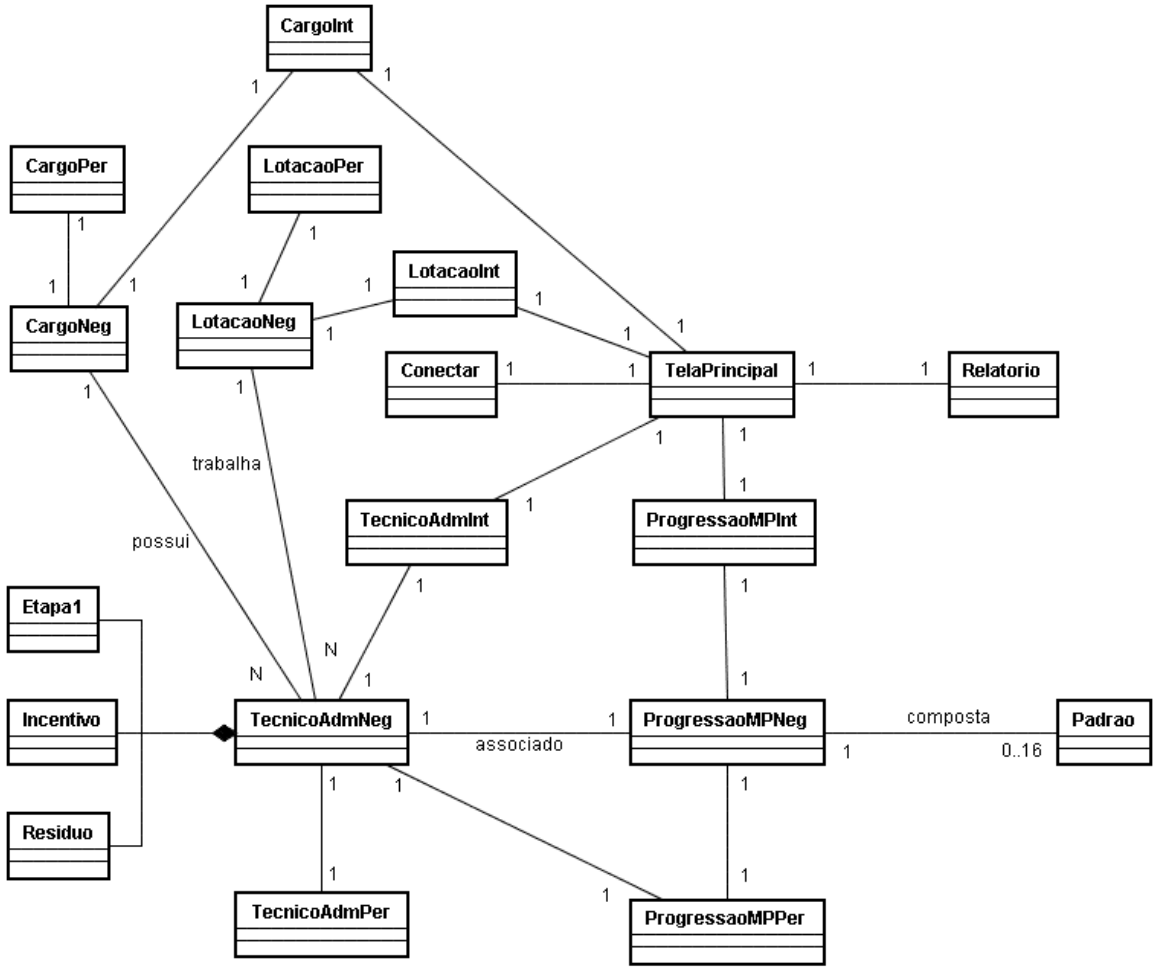


Figura 6.42 – Diagrama de Classes – Modelo de Projeto

6.4.2.3.2 Dicionário de Atributos e Métodos

Algumas das classes presentes no Diagrama de Classes do Modelo de Projeto não existem do Dicionário de Atributos e Métodos do Modelo de Análise, assim como há algumas classes idênticas e outras parcialmente idênticas. As tabelas apresentam o Dicionário de Atributos e Métodos das seguintes classes que sofreram alguma alteração ou que foram acrescentados no Modelo de Projeto:

- TecnicoAdmInt (Tabela 6.46);
- TecnicoAdmNeg (Tabela 6.47);
- TecnicoAdmPer (Tabela 6.48);
- Padrao (Tabela 6.49);
- ProgressaoMPInt (Tabela 6.50);
- ProgressaoMPNeg (Tabela 6.51);
- ProgressaoMPPer (Tabela 6.52);

Os Dicionários de Atributos e Métodos das classes TelaPrincipal, Conectar e Relatório são idênticos aos presentes no Modelo de Projeto do Produto de Software CPPD.

As classes relativas a Cargo, Lotação e Técnico Administrativo do Modelo de Projeto são semelhantes, pois elas possuem operações de incluir, alterar e excluir, além de atributos relativos ao banco de dados.

A classe TecnicoAdmNeg é semelhante à classe TecnicoAdm, diferenciando-se por possuir os atributos idCargo (utilizado como chave estrangeira no banco de dados e identificando o cargo relativo ao técnico-administrativo) e idLotacao (utilizado como chave estrangeira no banco de dados e identificando a lotação relativa ao técnico-administrativo), os métodos para atribuir e obter o valor do atributo e os métodos referentes à comunicação com a camada de persistência.

A classe TecnicoAdmNeg é semelhante à classe ProgressaoMP, diferenciando-se por possuir os métodos que comunicam com a camada de persistência.

Tabela 6.46 – Dicionário de Atributos e Métodos da Classe TecnicoAdmInt

Classe	**TecnicoAdmInt**				
Descrição	Responsável pela troca de mensagem entre o sistema e o usuário em relação às operações sobre os Técnicos Administrativos.				
Atributos					
Identificador	siape				
Descrição	número siape do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Identificador	tecnicoAdm				
Descrição	Um técnico administrativo do sistema				
TAD	Objeto TecnicoAdmNeg				
Métodos					
Identificador	ValidarDados				
Descrição Funcional	Verifica se os dados foram preenchidos corretamente				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	retorno		**TAD**	Booleano	

Tabela 6.47 – Dicionário de Atributos e Métodos da Classe TecnicoAdmNeg

Classe	**TecnicoAdmNeg**				
Descrição	Identifica e manipula informações sobre os técnicos administrativos.				
Atributos					
Identificador	idCargo				
Descrição	1 do cargo do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro

Identificador	idLotacao				
Descrição	Identificador da lotação onde o técnico administrativo exerce suas funções				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Métodos					
Identificador	setIdCargo				
Descrição Funcional	Seta o cargo do técnico administrativo				
Parâmetros de Entrada	idCargo		**TAD**	Inteiro	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getIdCargo				
Descrição Funcional	Obtém o cargo do técnico administrativo				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	idCargo		**TAD**	Inteiro	
Identificador	setIdLotacao				
Descrição Funcional	Seta a lotação do técnico administrativo				
Parâmetros de Entrada	idLotacao		**TAD**	Inteiro	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	getIdLotacao				
Descrição Funcional	Obtém a lotação do técnico administrativo				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	idLotacao		**TAD**	Inteiro	
Identificador	IncluirTecnicoAdm				
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, inserir um técnico administrativo no banco de dados.				
Parâmetros de Entrada	tecnicoAdm		**TAD**	Objeto TecnicoAdmNeg	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	VerificarExistencia				
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, verificar se determinado técnico administrativo existe.				
Parâmetros de Entrada	siape		**TAD**	Inteiro	
Parâmetros de Saída	retorno		**TAD**	Booleano	
Identificador	VerificarExistencia				
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, verificar se determinado técnico administrativo existe.				
Parâmetros de Entrada	nome		**TAD**	String	
Parâmetros de Saída	retorno		**TAD**	Booleano	

Identificador	ObterTecnicoAdm		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, obter um técnico administrativo no banco de dados.		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	tecnicoAdm	**TAD**	Objeto TecnicoAdmNeg
Identificador	AlterarTecnicoAdm		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, alterar os dados de determinado técnico administrativo no Banco de Dados.		
Parâmetros de Entrada	tecnicoAdm	**TAD**	Objeto TecnicoAdmNeg
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ExcluirTecnicoAdm		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, excluir um técnico administrativo no banco de dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.48 – Dicionário de Atributos e Métodos da Classe TecnicoAdmPer

Classe	**TecnicoAdmPer (classe estática)**		
Descrição	Responsável pelas operações sobre o Banco de Dados.		
Métodos			
Identificador	VerificarExistencia		
Descrição Funcional	Verifica se determinado técnico administrativo existe no Banco de Dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	retorno	**TAD**	Booleano
Identificador	VerificarExistencia		
Descrição Funcional	Verifica se determinado técnico administrativo existe no Banco de Dados		
Parâmetros de Entrada	nome	**TAD**	String
Parâmetros de Saída	retorno	**TAD**	Booleano
Identificador	IncluirTecnicoAdm		
Descrição Funcional	Inclui um técnico administrativo no Banco de Dados		
Parâmetros de Entrada	tecnicoAdm	**TAD**	Objeto TecnicoAdmNeg
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	AlterarTecnicoAdm		
Descrição Funcional	Altera os dados de determinado técnico administrativo no Banco de Dados		
Parâmetros de Entrada	tecnicoAdm	**TAD**	Objeto TecnicoAdmNeg
Parâmetros de Saída	nenhum	**TAD**	nenhum

Identificador	ExcluirTecnicoAdm		
Descrição Funcional	Exclui um técnico administrativo do Banco de Dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ObterTecnicoAdm		
Descrição Funcional	Obtém determinado técnico administrativo do banco de dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	tecnicoAdm	**TAD**	Objeto TecnicoAdmNeg
Identificador	ModificarPrxProgressaoMP		
Descrição Funcional	Modifica a data da próxima progressão do técnico administrativo		
Parâmetros de Entrada	proTecAdm, siape	**TAD**	Objeto ProgressaoMPNeg, Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum

Tabela 6.49 – Dicionário de Atributos e Métodos da Classe Padrao

Classe	**Padrao**				
Descrição	Identifica e manipula o padrão de progressão dos técnicos administrativos				
Atributos					
Identificador	id				
Descrição	Identificador do padrão				
Valores Possíveis	1 a 2147483647				
Tamanho	2	**Formato**	Números	**TAD**	Inteiro
Métodos					
Identificador	setId				
Descrição Funcional	Seta o identificador do padrão				
Parâmetros de Entrada	id		**TAD**	Inteiro	
Parâmetros de Saída	nenhum		**TAD**	nenhum	
Identificador	GetId				
Descrição Funcional	Obtém o identificador do padrão				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	id		**TAD**	Inteiro	

Tabela 6.50 – Dicionário de Atributos e Métodos da Classe ProgressaoMPInt

Classe	**ProgressaoMPInt**
Descrição	Responsável pela troca de mensagem entre o sistema e o usuário em relação às operações sobre as progressões por mérito profissional dos técnicos administrativos
Atributos	
Identificador	siape

Descrição	número siape do técnico administrativo				
Valores Possíveis	1 a 2147483647				
Tamanho	10	**Formato**	Números	**TAD**	Inteiro
Identificador	progressao				
Descrição	Progressão por mérito profissional de determinado técnico administrativo				
TAD	Objeto ProgressaoMPNeg				
Métodos					
Identificador	ValidarDados				
Descrição Funcional	Verifica se os dados foram preenchidos corretamente				
Parâmetros de Entrada	nenhum		**TAD**	nenhum	
Parâmetros de Saída	retorno		**TAD**	Booleano	

Tabela 6.51 – Dicionário de Atributos e Métodos da Classe ProgressaoMPNeg

Classe	**ProgressaoMPNeg**		
Descrição	Identifica e manipula informações sobre a progressão por mérito profissional dos técnicos administrativos		
Métodos			
Identificador	ModificarProgressao		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, alterar os dados de determinada progressão por mérito profissional no banco de dados.		
Parâmetros de Entrada	p, siape	**TAD**	Objeto ProgressaoMPNeg, Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum
Identificador	ObterProgressao		
Descrição Funcional	Comunica com a camada de persistência para, posteriormente, obter determinada progressão por mérito profissional no banco de dados.		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	p	**TAD**	Objeto ProgressaoMPNeg

Tabela 6.52 – Dicionário de Atributos e Métodos da Classe ProgressaoMPPer

Classe	**ProgressaoMPPer (classe estática)**		
Descrição	Responsável pelas operações sobre o Banco de Dados.		
Métodos			
Identificador	ModificarProgressao		
Descrição Funcional	Altera os dados de determinada progressão por mérito profissional no banco de dados		
Parâmetros de Entrada	p, siape	**TAD**	Objeto ProgressaoMPNeg, Inteiro
Parâmetros de Saída	nenhum	**TAD**	nenhum

Identificador	ObterProgressao		
Descrição Funcional	Obtém determinada progressão no Banco de Dados		
Parâmetros de Entrada	siape	**TAD**	Inteiro
Parâmetros de Saída	progressao	**TAD**	Objeto ProgressaoNeg

6.4.2.3.3 Diagramas de Seqüência

A seguir, são apresentados os seguintes Diagramas de Seqüência:

- Incluir Cargo (Figura 6.43);
- Alterar Cargo (Figura 6.44);
- Excluir Cargo (Figura 6.45);
- Incluir Técnico Administrativo (Figura 6.47);
- Alterar Técnico Administrativo (Figura 6.48);
- Excluir Técnico Administrativo (Figura 6.46);
- Consultar Técnico Administrativo (Figura 6.49);
- Modificar Progressão por Mérito Profissional (Figura 6.50);
- Consultar Progressão por Mérito Profissional (Figura 6.51);
- Emitir Relatório por Exercício (Figura 6.52).

Os diagramas para os casos de uso Manter Cadastro de Cargo e Lotação são semelhantes. Os Diagramas de Seqüência para os casos de uso de emissão de relatórios são semelhantes.

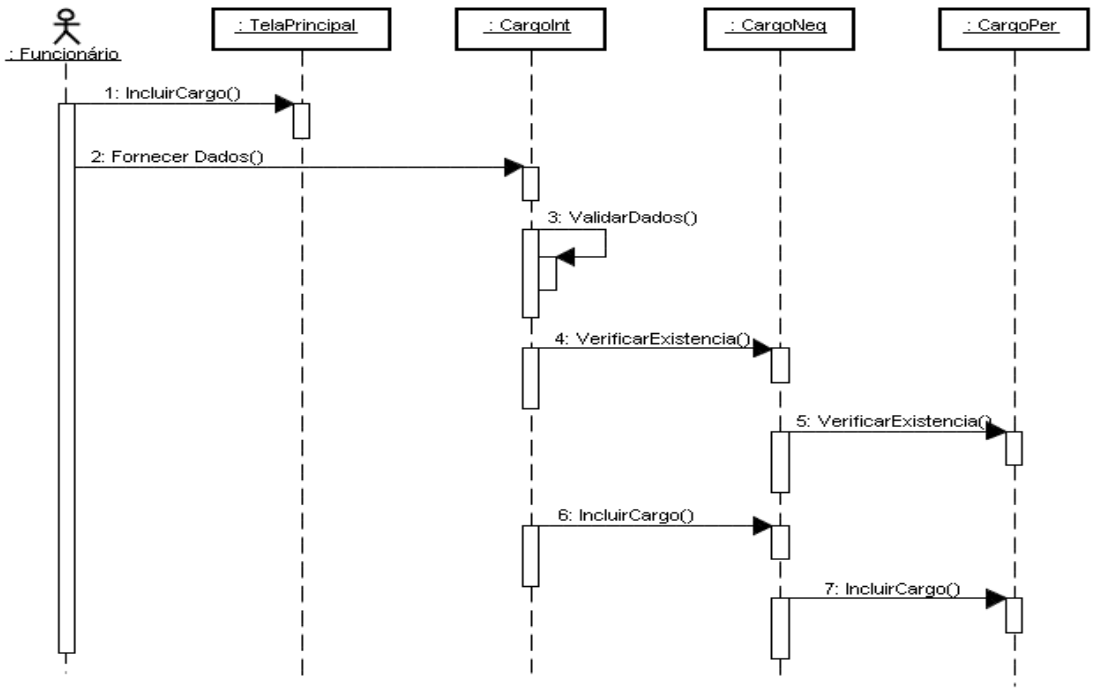


Figura 6.43 – Diagrama de Seqüência – Incluir Cargo

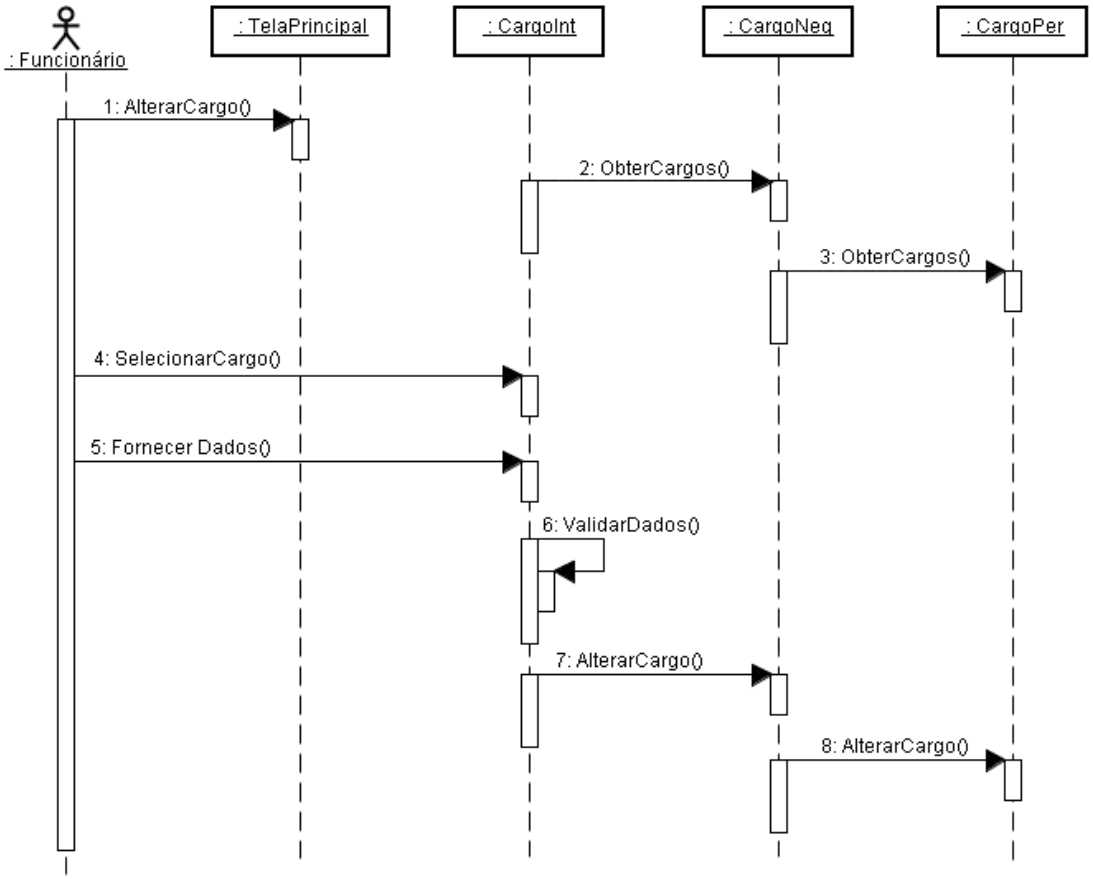


Figura 6.44 – Diagrama de Seqüência – Alterar Cargo

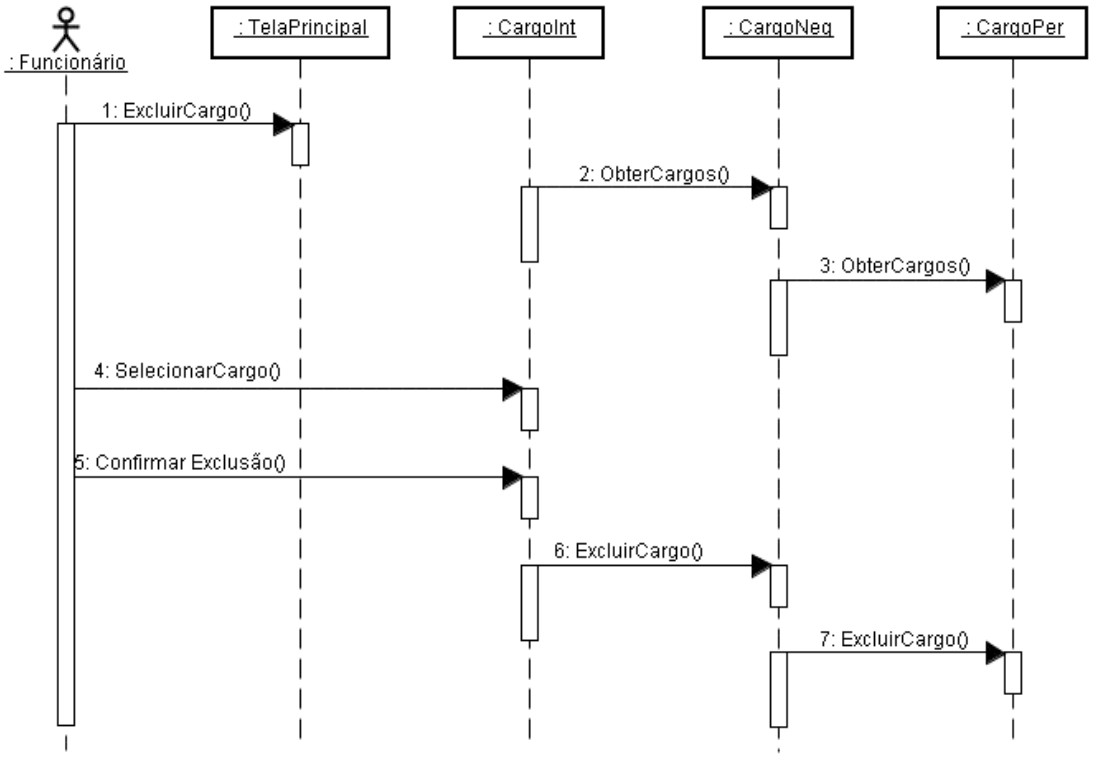


Figura 6.45 – Diagrama de Seqüência – Excluir Cargo

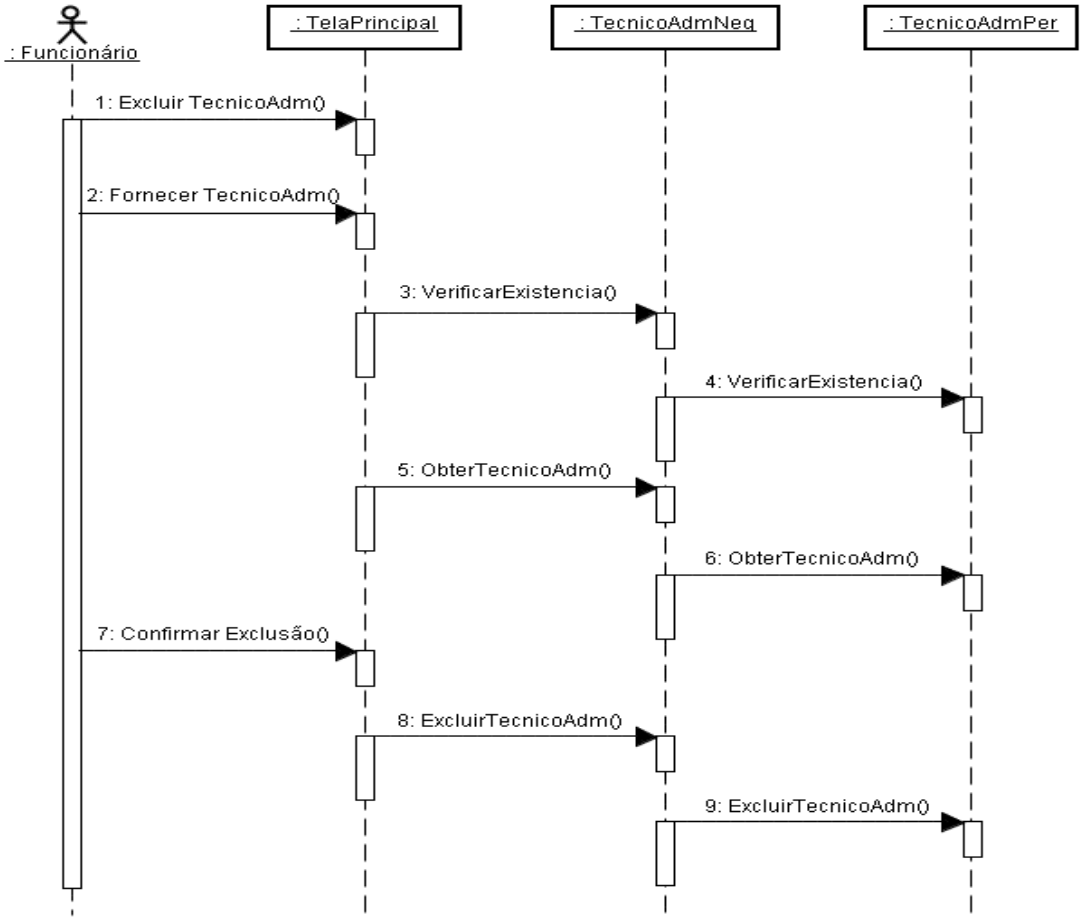


Figura 6.46 – Diagrama de Seqüência – Excluir Técnico Administrativo

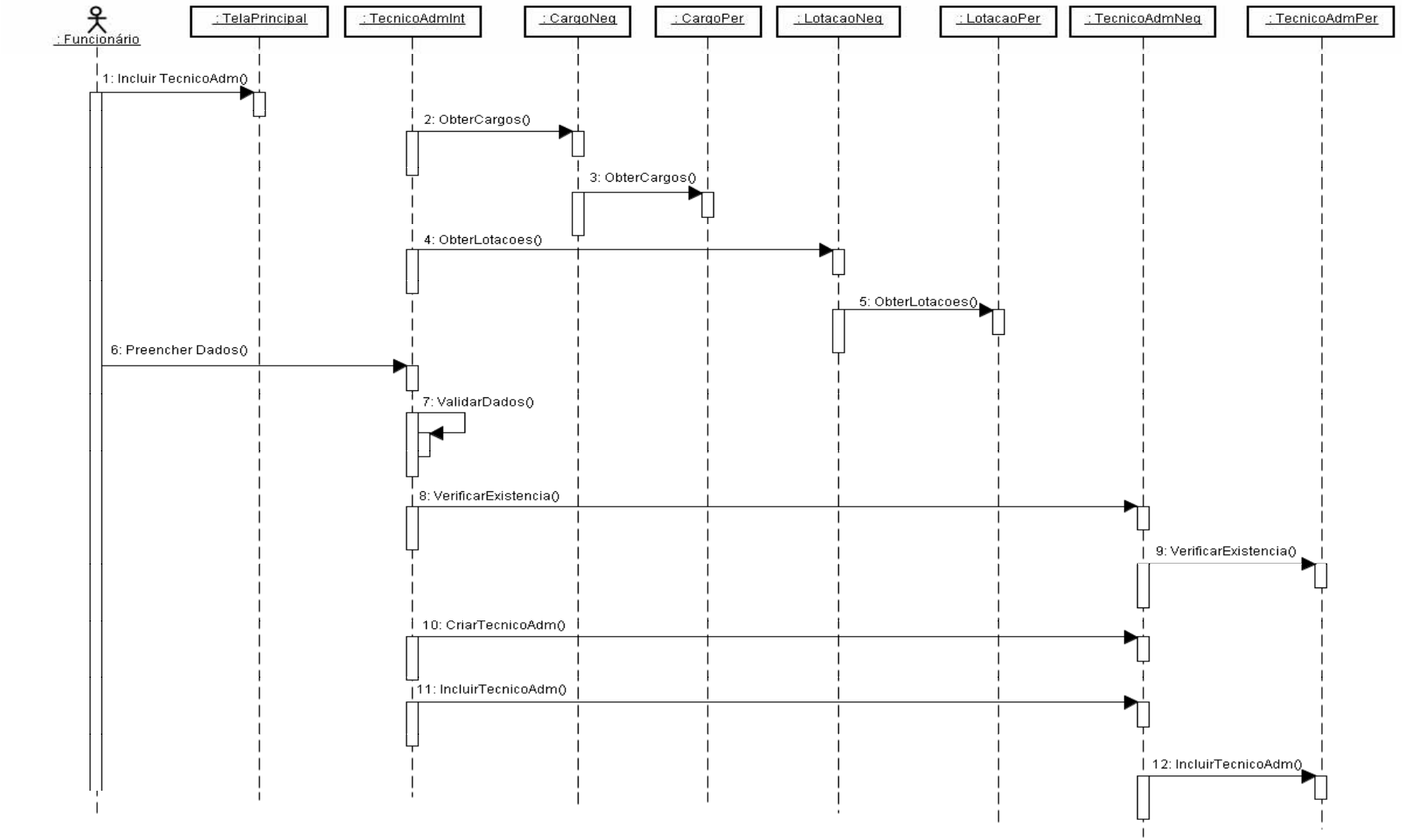


Figura 6.47 – Diagrama de Seqüência – Incluir Técnico Administrativo

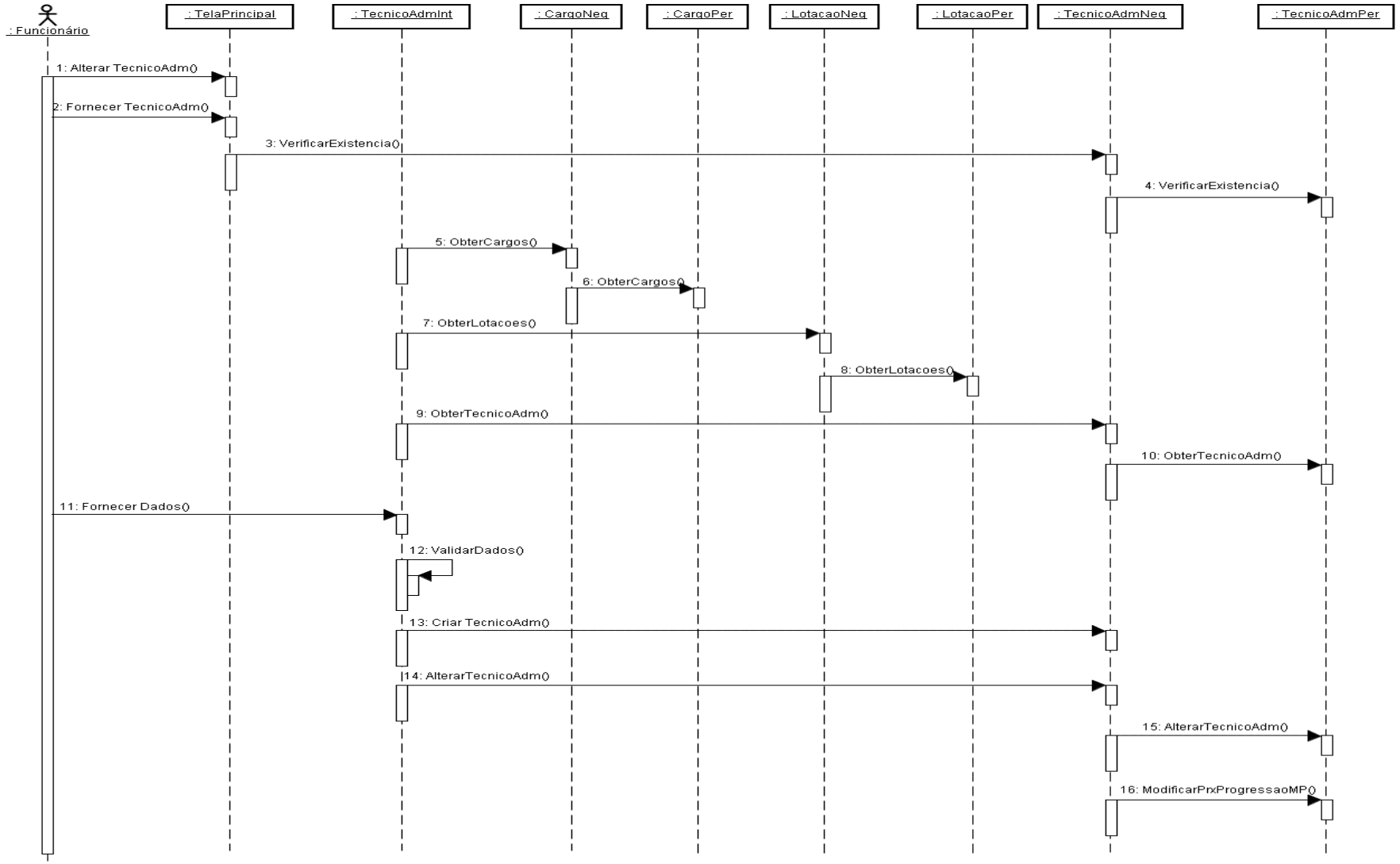


Figura 6.48 – Diagrama de Seqüência – Alterar Técnico Administrativo

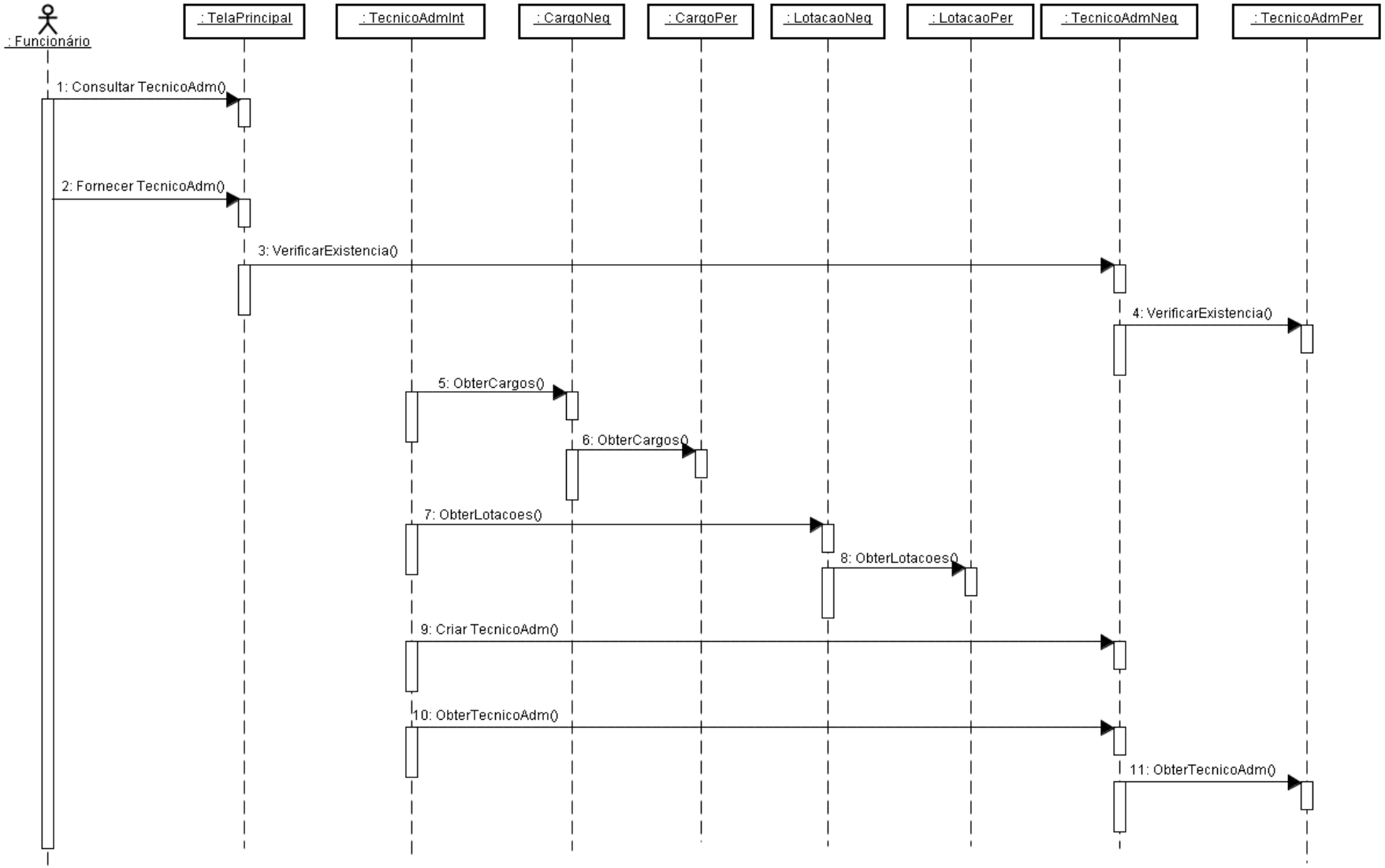


Figura 6.49 – Diagrama de Seqüência – Consultar Técnico Administrativo

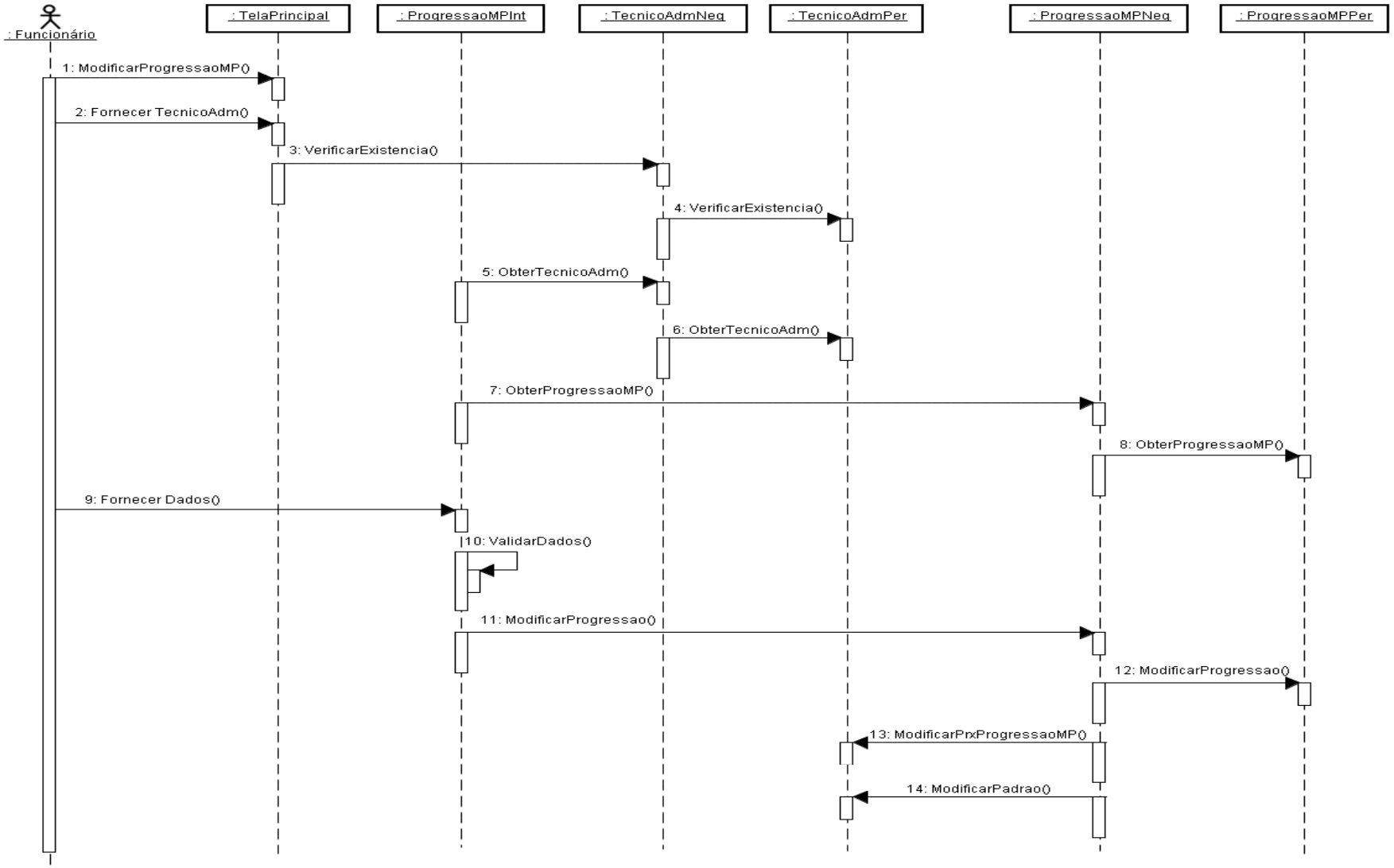


Figura 6.50 – Diagrama de Seqüência – Modificar Progressão por Mérito Profissional

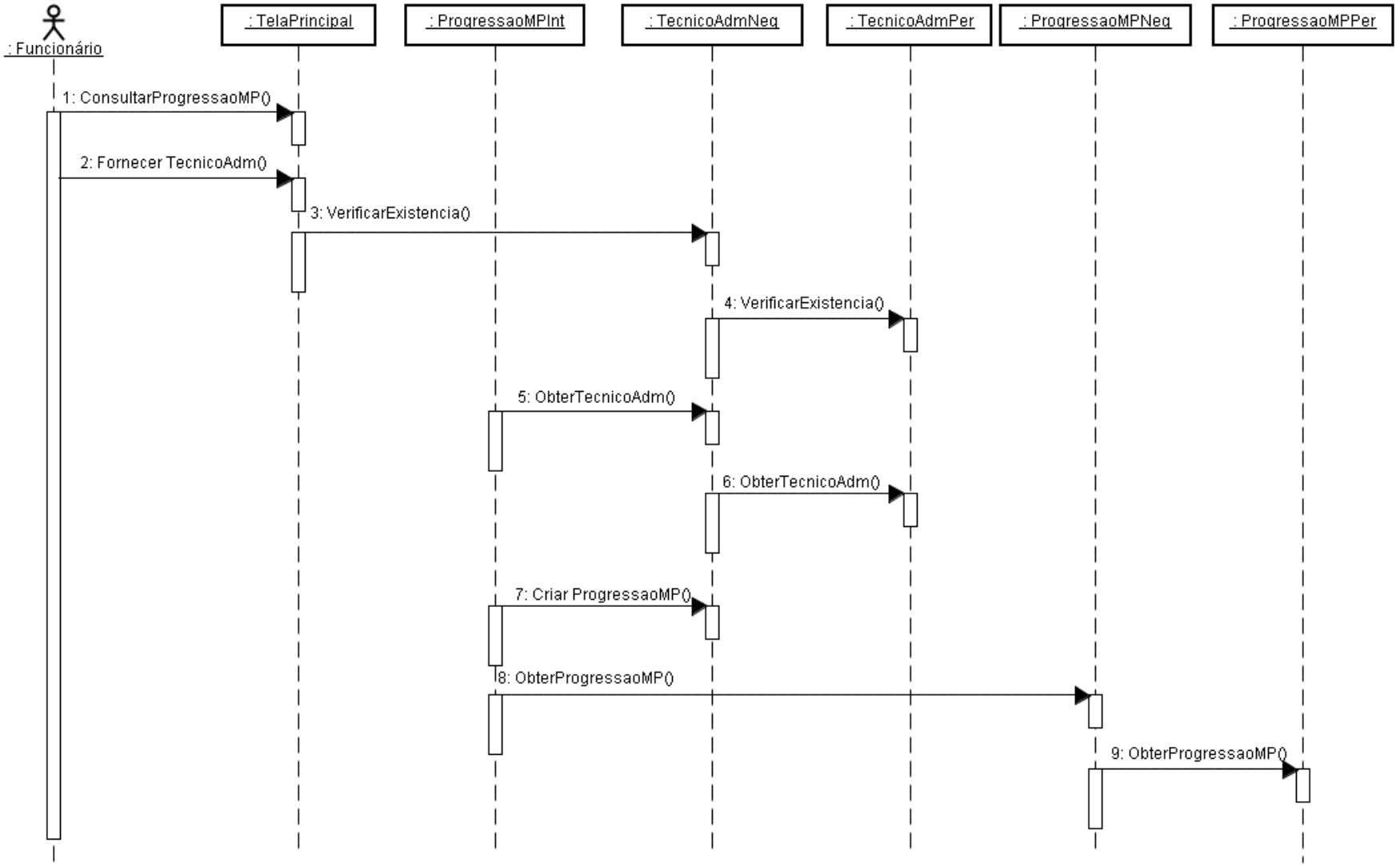


Figura 6.51 – Diagrama de Seqüência – Consultar Progressão por Mérito Profissional

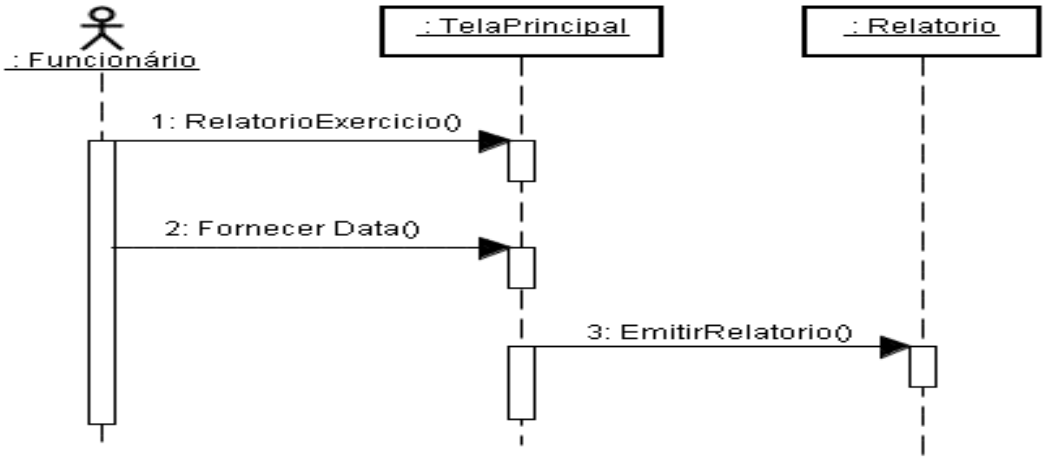


Figura 6.52 – Diagrama de Seqüência – Emitir Relatório por Exercício

6.4.3 Apresentação do Sistema

Nesta seção, é apresentado o produto de software descrevendo algumas situações de uso.

Mediante autenticação, a tela principal é exibida ao usuário que permite acesso a todos os menus do produto de software, sendo possível realizar as seguintes operações:

- Cadastro: incluir, alterar, excluir, consultar técnico administrativo;
- Progressão: modificar, consultar progressão por mérito profissional;
- Cargo: incluir, alterar, excluir cargo;
- Lotação: incluir, alterar, excluir lotação;
- Relatório: emissão de relatórios;
- Ajuda: informações sobre o sistema.

Através do menu Cargo, é possível executar a operação Incluir Cargo. A Figura 6.53 apresenta a tela de inclusão. Após preencher os dados e clicar em Incluir, é realizada a validação dos dados, verificação da existência do cargo e, por fim, os dados são incluídos no banco de dados.

A Figura 6.54 ilustra tela exibida ao selecionar a opção alterar técnico administrativo, dentro do contexto do caso de uso Manter Cadastro de Técnico Administrativo. Após informar o número siape ou nome do técnico administrativo desejado, é possível modificar seus dados através da tela de alteração. Seguindo os passos do Diagrama de Seqüência correspondente, verifica-se que a interface comunica-se com a lógica de negócio que, por sua vez, se comunica com a camada de persistência, obtendo as

informações do técnico administrativo presente no banco de dados. Após fornecer os novos dados e clicar no botão Alterar, novamente há comunicação entre as camadas e os dados são alterados.

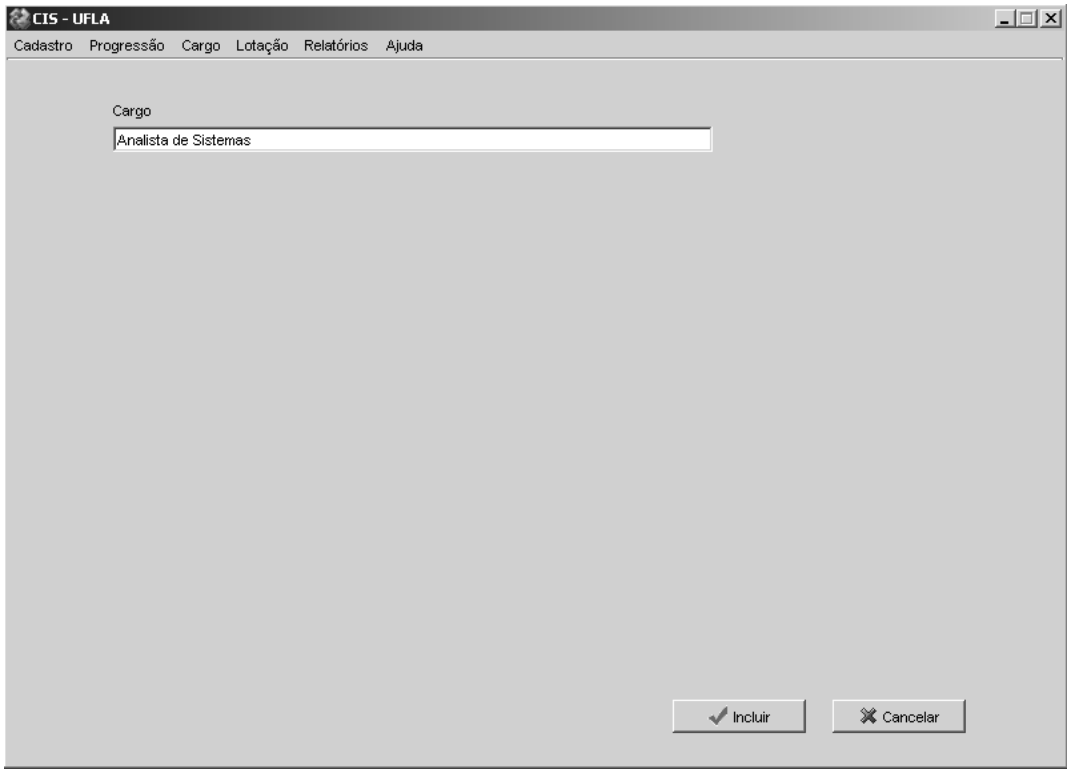


Figura 6.53 – Operação Incluir Cargo

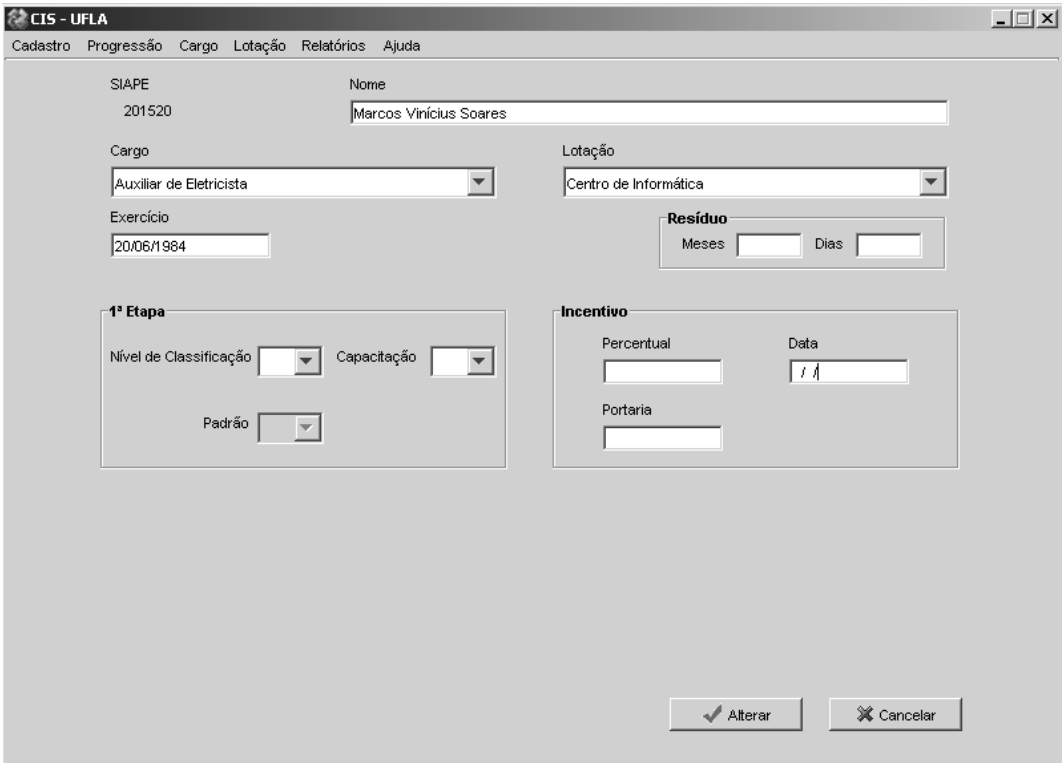


Figura 6.54 – Operação Alterar Técnico Administrativo

A progressão por mérito profissional dos técnicos administrativos é modificada através do menu Progressão. Fornecendo o número siape ou nome do técnico administrativo, a tela de progressão é exibida. Após informar os novos dados, ocorre uma validação e, posteriormente, os dados são alterados no banco de dados. A Figura 6.55 representa a tela para modificar a progressão por mérito profissional.

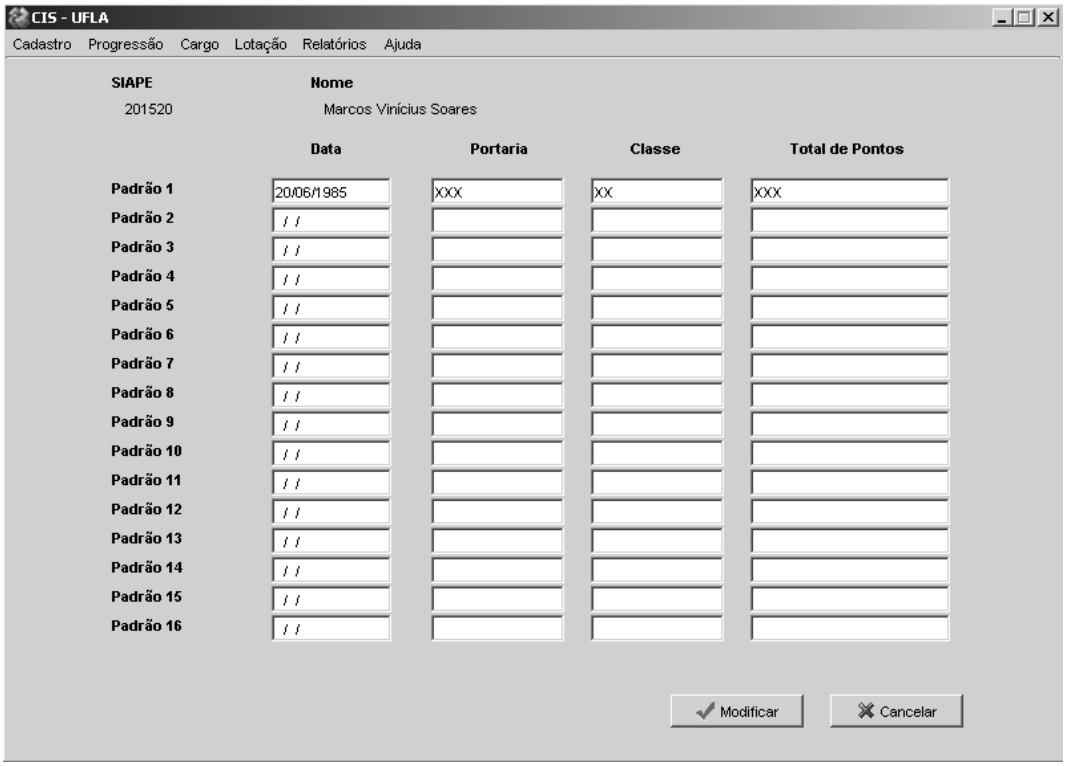


Figura 6.55 – Operação Modificar Progressão por Mérito Profissional de Técnico Administrativo

A emissão de relatórios por cargo, por lotação, por exercício e por progressão por mérito profissional do mês é possível através do menu Relatórios. A Figura 6.56 ilustra a tela representando a operação emitir relatório por exercício. No caso, foram selecionados o mês de Agosto e o ano de 2004.

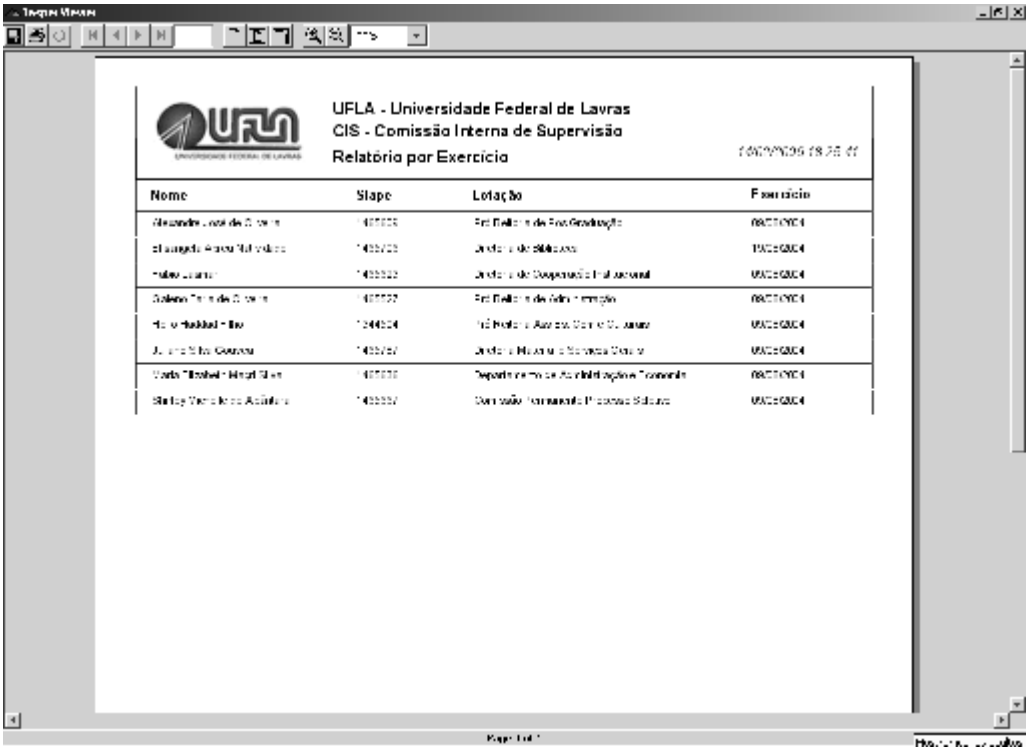


Figura 6.56 – Operação Emitir Relatório por Exercício

6.5 Considerações Finais

Neste capítulo foi apresentado o desenvolvimento dos produtos de software CPPD e CIS, descrevendo detalhadamente as fases a que foram submetidos. Ambos são utilizados pela Diretoria de Recursos Humanos e facilitam o cadastro e o controle dos processos funcionais dos docentes e técnicos administrativos da Universidade Federal de Lavras.

7 CONSIDERAÇÕES FINAIS

O presente trabalho apresentou uma proposta de manutenção para o produto de software CPPD e uma proposta de desenvolvimento para o produto de software CIS, fornecendo, em seu término, os dois produtos de software.

7.1 Conclusão

A manutenção do produto de software CPPD foi realizada com base nas técnicas de Engenharia Reversa e Reestruturação. Sendo assim, foi possível recuperar informações de projeto perdidas durante a fase de desenvolvimento e documentar o real estado do produto de software. Em seguida o produto de software foi reestruturado, sendo novamente desenvolvido utilizando tecnologia mais avançada. O uso dessas técnicas foi essencial para realizar a manutenção do produto de software CPPD.

O desenvolvimento dos produtos de software foi baseado no Modelo Interativo e Incremental, fornecendo, ao longo do tempo, incrementos dos produtos de software ao cliente. O cliente não precisou esperar até que os produtos de software estivessem totalmente desenvolvidos para usufruir de suas funcionalidades e, com base neste modelo, o risco de erros foi reduzido, pois, como os produtos foram entregue em partes menores, o gerenciamento e tratamento de riscos foi facilitado.

Os produtos de software CPPD e CIS demonstraram-se eficazes, atendendo às especificações propostas. O desempenho da aplicação utilizando um servidor Web correspondeu às expectativas esperadas, recuperando informações armazenadas no SGBD, obtendo também uma grande portabilidade entre diferentes plataformas de software e hardware, possibilitada pela utilização da tecnologia Java. A documentação gerada para ambos os produtos de software irá auxiliar futuros processos de manutenção.

7.2 Contribuições

Os produtos de software CPPD e CIS provêem de forma fácil e rápida o acesso a dados, informações e conhecimento relevantes para a realização das funções desempenhadas pela Diretoria de Recursos Humanos da Universidade Federal de Lavras.

Podemos citar ainda, outros benefícios adivinhos da utilização dos produtos de software:

- Melhoria no processo de extração de informações estatísticas e gerenciais, contribuindo com a geração de indicadores e facilitando o acesso aos dados dos servidores da universidade;
- Aumento de disponibilidade e agilidade atendendo as demandas de informação;
- Melhoria na qualidade da informação gerada, contribuindo com a eliminação das redundâncias e oferecendo transparência e confiabilidade aos dados;
- Viabilização da execução dos sistemas remotamente, para que, futuramente, outros órgãos e departamentos da universidade tenham acesso aos dados.

No decorrer do desenvolvimento deste trabalho incrementei, de forma considerável, meu conhecimento sobre Engenharia de Software, em especial sobre Desenvolvimento e Manutenção de Produtos de Software. Através do estudo aprofundado de processos, técnicas e modelos de desenvolvimento e manutenção, sinto-me apto a desenvolver e manter produtos de software visando alta qualidade, eficácia e eficiência. Além do mais, acredito que este trabalho possa ser utilizado, por muitos, como um guia para desenvolvimento e manutenção de produtos de software.

7.3 Trabalhos Futuros

Seria interessante que outros órgãos e departamentos da Universidade Federal de Lavras possuíssem acesso ao produto de software. Para isso, seria necessário o desenvolvimento de um esquema de autenticação mais bem elaborado, restringindo o acesso dos usuários a visualizarem somente informações relativas a tais órgãos e/ou departamentos. Além do mais, outros dados dos servidores poderiam ser cadastrados no sistema e mais relatórios poderiam ser desenvolvidos, incrementando, desta forma, a funcionalidade dos produtos de software.

Se for de interesse de outras Instituições de Ensino Superior (IES) adquirir os produtos de software com o intuito de melhor administrar os dados relativos a pessoal, poderia ser disponibilizada uma versão em meio magnético dos produtos CPPD e CIS.

Outro fator importante é o de que a evolução dos produtos de software deve acompanhar a evolução da tecnologia utilizada em seu desenvolvimento e manutenção. Portanto, é necessário ficar atento ao lançamento de novas versões destas ferramentas, adaptando os produtos de software sempre que possível, visando maior eficiência, segurança e confiabilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

ANSI/IEEE – An American National Standart IEEE Standart. **Glossary of Software Engineering Terminology.** IEEE Transaction on Software Engineering. 1983

ARTHUR, L. J. Software Evolution. **The Software Maintenance Challenge**. John Wiley & Sons, 1988.

BECK, K. (1999). **Embracing change with extreme programming**. IEEE Computer, 32(2), p. 70-78. (Cap. 3)

BENNETT K. H., RAJLICH V.T. **Software Maintenance and Evolution: A Roadmap**. In A Finkelstein (ed.) The Future of Software Engineering, ACM Press, 2000.

BENNETT, K. H., CORNELIUS, B., MUNRO, M., ROBSON D. **Software Maintenance**. In J. A. McDermid, editor, Software Engineer's Reference Book. Butterworth-Heinemann, 1991.

BOEHM, B. W. **Software Enginnering**. IEEE Transactions on Computer, 25(12), pp. 1226-1241, 1976.

BOEHM, B.W.(1988). **A spiral model of software development and enhancement**. IEEE Computer, 21(5), p. 61-72. (Caps. 3 e 4)

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Addison-Wesley, 1999. 482p.

BOOCH, Grady. **Object-Oriented Analysis and Design with Applications**. 2nd Edition. Benjamim/Cummings Publishing Company Inc, 1994.

CHAPIN, Ned. **Software Maintenance Types – A Fresh View**. IEEE International Conference on Software Maintenance (ICSM'00), p.1063, 2000.

BRADAC, M., PERRY, D., VOTTA, L. **Prototyping a Proccess Monitoring Experiment**. IEEE Trans. Software Engineering, vol. SE-20, no. 10, October 1994, pp. 774-784.

BROOKS, F. **The Mythical Man-Month**, Addison-Wesley, 1975.

CAPRETZ, Luiz F. & CAPRETZ, Miriam A. M. **Object-Oriented Software: Design and Maintenance**. University of Pittsburgh, USA. Series Editor S K Chang: Series on Software Engineering and Knowledge Engineering, Vol. 6, 1996.

CHIKOFSKY, E. J. & CROSS II, J. II. **Reverse Engineering and Design Recovery: Taxonomy**. IEEE Software, v. 7, n.1, 1990,p.13-17.

COUSIN, L. & COLLOFELO, J. S. **A task-based approach to improving the software maintenance process**. International Conference on Software Maintenance (ICSM) – IEEE Orlando, Florida nov., 9-12, 1992.

DEITEL, H.M. e DEITEL, P.J. **Java How to Program, 4th Edition**. Prentice Hall, 2002.

DEMARCO, T. **Software development: State of the art vs. state of the practice (1993)**. In: Why does Software Cost so Much? New York: Dorset House, 1st. ed., 1995.

DIAS, Márcio Greyck Batista, ANQUENTIL, Nicolas, OLIVEIRA, Káthia. **Organizing the Knowledge Used in Software Maintenance Projects**. Luzern, Switzerland: LSO´03 Workshop on Learning Software Organizations. In: PROCEEDINGS OF THE 2ND KONFERENZ PROFESSIONELLES WISSENSMANAGEMENT ERFAHRUNGEN UND VISIONEN. May, 2003.

EINSENMANN, A. L. K. & MARTINS, R. A. G. C. **Tecnologia da Computação. Curso preparatório para o exame do POSCOMP**. pp. 71-73. 2005.

ELMASRI, R & NAVATHE, S. B. **Sistemas de Banco de Dados : Fundamentos e Aplicações**. Tradução Teresa Cristina Padilha de Souza; revisão técnica Sérgio da Costa Côrtes. 3. ed. Rio de Janeiro: LTC - Livros Técnicos e Científicos, 2002. 837 p.

ERIKSSON, H. E.; PENKER, M. **UML Toolkit**. New York : John Wiley, 1998.

FOSTER, J. R., JOLLY, A. E. P., NORRIS, M. T. **An overview of software maintenance**. British Telecom Technology Journal, 7(4), pp. 37-46, 1989.

FURLAN, José Davi. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

GEYER. **Análise de Ferramentas de Modelagem UML Gratuitas**. 2005 Disponível em <http://www.inf.ufrgs.br/procpar/disc/cmp167/trabalhos/sem2001-1/T1/alex/>. Acesso em 03 nov. 2005.

GILB, T. **Principles of Software Engineering Management**, Addison-Wesley, 1988.

HANNA, M. **Farewell to Waterfalls**, Software Magazine, May 1995, pp. 38-46.

HEPTAGON. **Heptagon: Artigo - FDD.** 2005. Disponível em <http://www.heptagon.com.br/artigos/fdd-udp-xp.htm>. Acesso em 02 nov. 2005.

IREPORT. **iReport - OpenSource Java Reporting Tool**. 2005. Disponível em <http://ireport.sourceforge.net/>. Acesso em 31 out. 2005.

JACOBSON, I. **Objectory is the unified process**. Component Strategies, v.1, n. 10, Apr., 1998 p. 67-72.

JACOBSON, I., BOOCH, G., RUMBAUGH, J. **The unified software development process**. Reading: Addison-Wesley, 1999.

JASPER. **JasperReports – Home**. 2005. Disponível em <http://jasperreports.sourceforge.net/>. Acesso em 31 out. 2005.

JAVA**. Java 2 Platform, Standard Edition (J2SE)**. 2005. Disponível em <http://java.sun.com/j2se/index.jsp>. Acesso em 30 out. 2005.

LIENTZ B. P., SWANSON E. B. **Software Maintenance Management**. Addison Wesley, Reading, MA, 1980.

MARTIN, J. & MCCLURE, C. **Software Maintenance – The Problem and Its Solutions**. Prentice Hall, 1983.

MCDERMID, J. & ROOK, P., **Software Development Proccess Models**, in Software Engineer's Reference Book, CRC Press, 1993, pp. 15/26-15/28.

MILLS, H. D., DYER, M. *et al* (1980). **The management of software engineering**. IBM Sys J., 24(2), p 414-477. (Cap.3)

MYSQL. **MySQL AB :: Why MySQL?**. 2005. Disponível em <http://www.mysql.com/why-mysql/>. Acesso em 31 out. 2005.

NIERSTRASZ, O., GIBBS, S., TSICHRITZIS, D. **Component-Oriented Software Development**, CACM, vol. 35, no. 9, September 1992, pp. 160-165.

PFLEEGER, S. L. **Software Engineering: Theory and Practice**. 2nd Edition. New Jersey: Prentice Hall, 2001.

PRESSMAN, Roger S. **Engenharia de Software**. – 5.ed. – Rio de Janeiro: McGraw-Hill, 2002.

PRESSMAN, Roger S. **Software Engineering**. 5th Edition. p.225-241, McGraw Ilill, 2001.

RAJLICII, V. & SRIDIIAR, R.. **VIFOR 2: A tool for browsing and documentation**. International Conference on Software Maintenance (ICSM) – IEEE Monterey, California Nov., 4-8, 1996.

ROYCE, W. W. **Managing The Development of Large Software Systems: Concepts and Techniques**, Proc. WESCON, August 1970.

SCHNEIDEWIND, N. F. **Introduction to the Special Section of Software Maintenance. The State of Software Maintenance**. IEEE Transactions on Software Engineering, 1987. pp.303-310.

SILVA, J. C. & DÍSCOLA, S. L. **Processos de Planejamento da Reengenharia de Software Apoiados por Princípios de Usabilidade**. Universidade Federal de São Carlos. São Carlos, São Paulo, 2003.

SOMMERVILLE, Ian. **Engenharia de software/ Ian Sommerville**; tradução André Maurício de Andrade Ribeiro; revisão técnica Kechi Hirama. -- São Paulo: Addison Wesley, 2003.

STAIR, Ralph M. **Princípios de sistemas de informação**. Ralph M. Stair, George W. Reynolds; tradução Alexandre Melo de Oliveira; revisor técnico: Luiz Augusto Pereira de Andrade Figueira**.** Rio de Janeiro: LTC, 2002. 496 p. Traducao de: Principles of information systems.

SUN. **Sun Microsystems**. 2005. Disponível em <http://www.sun.com/download/products.xml?id=3fb913e7>. Acesso em 31 out. 2005.

TILLEY, Scott R. **Documenting-in-the-large vs. Documenting-in-the-small**, Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: distributed computing – Volume 2 (CASCON'93), Toronto, Ontario, Canada, 1993,p.1083-1090.

WONG Kenny, TILLEY, Scott R., MÜLLER, Hausi A., STOREY Margaret-Anne D. **Structural Redocumentation: A Case Study**, IEEE Software, Janeiro, 1995,p.46-54.