

ESCOLA SUPERIOR ABERTA DO BRASIL – ESAB
PÓS-GRADUAÇÃO LATO SENSU DE ENGENHARIA DE SISTEMAS

ANDRÉ DE MORAES MARTINS

SISTREINA

Sistema de controle de treinamentos

RIO DE JANEIRO – RJ

2009

ANDRÉ DE MORAES MARTINS

SISTREINA

Sistema de controle de treinamentos

Monografia apresentada à ESAB – Escola
Superior Aberta do Brasil, sob orientação
da Professora Beatriz Christo Gobbi

RIO DE JANEIRO – RJ

2009

ANDRÉ DE MORAES MARTINS

SISTREINA

Sistema de controle de treinamentos

Aprovada em de de 200....

RIO DE JANEIRO – RJ

2009

A todos que colaboraram com a conclusão do curso...

Ao meu filho, elemento central para minha evolução...

Resumo

Este trabalho acadêmico teve como objetivo desenvolver um sistema de controle de treinamento para os órgãos públicos. A demanda é vivenciada pelo próprio autor. O levantamento de requisitos foi cuidadoso e com vários servidores, principalmente do setor de recursos humanos. Sistemas semelhantes foram analisados, porém não foi encontrado nenhum que atendesse ao documento de visão. As melhores metodologias foram analisadas, dissecadas e implementadas. As tecnologias abordadas são gratuitas e de ampla difusão, apoiadas no paradigma da orientação a objetos. O processo de desenvolvimento, a linguagem e a análise possibilitarão manutenções evolutivas sem muito esforço para compreensão do código. A implantação real do sistema trará ganhos diretos para o serviço público.

Sumário

Introdução.....	6
Palavras-chave	6
Exposição do Assunto.....	6
Problema de Pesquisa.....	6
Justificativa para a escolha do tema.....	7
Objetivo geral.....	7
Objetivo específico.....	8
Delimitação do trabalho.....	8
Metodologia de Pesquisa.....	8
CAPÍTULO I – Estudos das Tecnologias e Metodologias.....	9
I.1 Gerência de Projeto de sistemas.....	9
I.2 Processo de Desenvolvimento de Software.....	12
I.2.1 O RUP.....	19
I.3 UML.....	30
I.4 JAVA.....	33
I.5 Banco de Dados Relacionais.....	36
I.6 MySQL.....	44
I.7 Arquitetura Web (tres camadas).....	47
I.7.1 Arquitetura de uma camada.....	47
I.7.2 Arquitetura de duas camadas.....	48
I.7.3 Arquitetura de três camadas.....	48
CAPÍTULO II - Comparação com outros sistemas.....	49
Conclusão.....	52
Referências bibliográficas.....	53
Anexo – Diagramas da UML e Gráfico de Gantt.....	54

INTRODUÇÃO

PALAVRAS-CHAVE

Avaliação Acadêmica Sistemas Web Controle de Treinamentos.

EXPOSIÇÃO DO ASSUNTO

Este trabalho busca analisar e utilizar as melhores práticas para o desenvolvimento de um sistema de controle de treinamentos, tratando assim de um problema vivenciado pelos servidores dos órgãos públicos.

PROBLEMA DE PESQUISA

Os servidores públicos, ao contrário dos trabalhadores da iniciativa privada, necessitam de treinamento não só para engrandecimento profissional, mas para possibilitar progressões em seus planos de carreira.

Como possibilitar um controle rápido e seguro dos treinamentos efetuados pelo servidor ? Tal conhecimento é tão importante para o interessado quanto para o setor de recursos humanos do órgão, o qual ainda necessita controlar outros fatores como qualidade e idoneidade das empresas prestadoras de serviço.

JUSTIFICATIVA PARA A ESCOLHA DO TEMA

A crescente ampliação e propagação das metodologias de desenvolvimento de softwares possibilita cada vez mais a produção de programas robustos, portáteis e de grande escalabilidade.

A aplicação de uma tecnologia moderna, voltada para atender questões clássicas da sociedade, é um fator extremamente motivador para o desenvolvimento de qualquer sistema.

Uma das necessidades dos servidores públicos, vivenciada pessoalmente pelo autor desta monografia, incentivou a criação deste trabalho acadêmico. Sua aplicação proporcionará grande economia para a administração pública de duas maneiras: diretamente, pois aumentará a eficiência do controle dos treinamentos por parte dos setores de recursos humanos, proporcionando maior economia e transparência; e indiretamente, uma vez que ampliará a satisfação dos servidores, os quais terão simples acesso ao seu histórico e poderão rapidamente propor melhorias para suas tarefas.

OBJETIVO GERAL

Estudar a criação de um sistema bem documentado, tanto lógico quanto conceitualmente, para a web, que solucione o problema da pesquisa.

OBJETIVO ESPECÍFICO

Analisar o desenvolvimento de um sistema de controle de treinamento que atenda tanto a necessidade dos servidores quanto a necessidade do setor de recursos humanos dos órgãos públicos, tendo como a base as melhores práticas de desenvolvimento de software.

DELIMITAÇÃO DO TRABALHO

Conceituar e comparar as tecnologias e metodologias utilizadas na produção de um software, mostrando os artefatos gerados pelo projeto, análise e implementação do sistema de controle de treinamentos como exemplo prático.

METODOLOGIA DE PESQUISA

Utilizou-se a metodologia aplicada, valendo-se de pesquisa bibliográfica para o estudo das ferramentas de apoio ao desenvolvimento, entrevistas com quatro servidores dos recursos humanos, no período de 29 de setembro de 2008 a 03 de outubro de 2008, para levantamento de requisitos e comparações com sistema atualmente existentes, os quais buscam atender o objetivo do trabalho (ou parte dele).

CAPÍTULO I – ESTUDOS DAS TECNOLOGIAS E METODOLOGIAS

I.1 GERÊNCIA DE PROJETO DE SISTEMAS

Um projeto de desenvolvimento de um sistema é um empreendimento único que deve ser realizado dentro de um prazo pré-estabelecido com os recursos e tecnologias disponíveis. Tem como objetivo atingir a meta estabelecida pelo solicitante do projeto, envolvendo um certo grau de incerteza na realização (MARTINS, 2002).

A gerência do projeto em questão possibilitou estimar o tempo necessário para a entrega do sistema, considerando-se os prazos estipulados. Além disso, organizou a seqüência de tarefas para a implementação.

Como uma área de estudo, a gerência de projetos é a aplicação de conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas para atingir um conjunto de objetivos pré-definidos. O conhecimento e as práticas da gerência de projetos são mais bem descritos em termos de seus processos componentes. Esses processos podem ser classificados em cinco grupos de processo (iniciação, planejamento, execução, controle e encerramento) e nove áreas de conhecimento (gerência de integração, gerência de escopo, gerência de tempo, gerência de custo, gerência de qualidade, gerência de recursos humanos, gerência de comunicações, gerência de riscos e gerência de aquisições).

Reduzida à sua forma mais simples, a gerência de projetos é a disciplina de manter os riscos de fracasso em um nível tão baixo quanto necessário durante o ciclo de

vida do projeto. O risco de fracasso aumenta de acordo com a presença de incerteza durante todos os estágios do projeto, como o surgimento de um fator externo desconhecido. Segundo Eduardo Gorges (2007), um dos fatores críticos de sucesso de um projeto é a gerência de escopo. Esta considera que conforme as atividades evoluem, o cliente demanda novas entregas. Tal evento é normal, mas as atividades devem ser replanejadas e novos prazos e custos devem ser acordados entre todos os interessados.

A gerência de projetos é frequentemente entregue a um indivíduo intitulado gerente de projeto. Idealmente, esse indivíduo raramente participa diretamente nas atividades que produzem o resultado final. Ao invés disso, o gerente de projeto trabalha para manter o progresso e a interação mútua progressiva dos diversos participantes do empreendimento, de modo a reduzir o risco de fracasso do projeto. Tal definição não se aplicou a esta obra, por obrigar a autoria de um único indivíduo, o qual foi gerente, analista e programador do projeto.

O gerenciamento de projetos tenta adquirir controle sobre três variáveis: tempo, custo e escopo. Algumas literaturas definem como quatro variáveis, sendo qualidade a quarta. Pode-se observar que esta é plenamente aplicável ao sistema em questão.

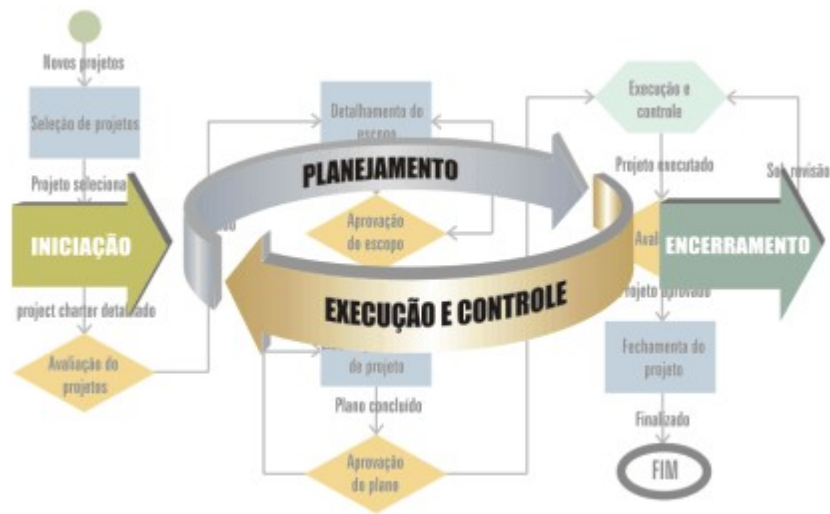


Figura 1 – Representação da Gerência de Projetos

A etapa de iniciação do projeto estabeleceu o objetivo desta obra, exposta na introdução. A etapa de planejamento, a qual baseou-se na data limite de entrega do projeto, criou o cronograma de atividades a seguir:

Tarefa	Início	Fim	Duração (dias úteis)
1- AnteProjeto	21/09/2008	23/09/2008	3d
2- Análise do sistema atual	24/09/2008	26/09/2008	3d
3- Levantamento de Requisitos	29/09/2008	10/10/2008	10d
3.1- Entrevista com usuário	29/09/2008	03/10/2008	5d
3.2- Comparação com o sistema atual	06/10/2008	09/10/2008	4d
3.3- Revisão dos requisitos	10/10/2008	10/10/2008	1d
4- Objetivo e Mini-Mundo	13/10/2008	15/10/2008	3d
5- Revisão do objetivo e mini-mundo	16/10/2008	17/10/2008	2d
6- Diagramas da UML	20/10/2008	11/11/2008	17d
6.1- Diagrama de Casos de Uso	20/10/2008	22/10/2008	3d
6.2- Diagrama de Classes	23/10/2008	24/10/2008	2d
6.3- Detalhamento dos casos de uso	27/10/2008	29/10/2008	3d

6.4- Diagrama de atividades	30/10/2008	31/10/2008	2d
6.5- Diagrama de Estados	03/11/2008	03/11/2008	1d
6.6- Diagrama de Sequencia	04/11/2008	06/11/2008	3d
6.7- Revisão dos Diagramas	07/11/2008	11/11/2008	3d
7- Diagrama de Entidade-Relacionamento	12/11/2008	14/11/2009	3d
8- Definição do layout	17/11/2008	17/11/2008	1d
9- Prototipagem das telas	18/11/2009	21/11/2009	4d
10- Definição do SGBD	24/11/2008	24/11/2008	1d
11- Criação da base de dados	25/11/2008	28/11/2008	4d
12- Definição e estudo da linguagem, IDE e Frameworks	01/12/2008	05/12/2008	5d
13- Codificação	08/12/2008	30/01/2009	35d
14- Testes	01/02/2009	27/02/2009	17d
15- Manuais	02/03/2009	06/03/2009	5d
16- Implantação e Treinamento	09/03/2009	14/03/2009	5d
Total de dias úteis para o projeto:			118d

Quadro 1 – Cronograma do Projeto

O paralelismo de atividades não pôde ser implementado por indisponibilidade de pessoal, porém seria factível. O gráfico de Gantt foi incluído nos anexos.

I.2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Um processo de desenvolvimento de software é um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de software. Logo, a adoção de um foi extremamente essencial para a conclusão do projeto. Segundo Michel Soares (2007), as atividades que contribuem para a produção de um software geram um produto que reflete a forma como todo o processo foi conduzido.

É estudado dentro da área de Engenharia de Software, sendo considerado um dos principais mecanismos para se obter software de qualidade e cumprir corretamente os contratos de desenvolvimento.

As principais atividades são: Análise de Requisitos, Especificação, Arquitetura, Implementação, Testes, Documentação, Suporte, Treinamento e Manutenção.

Análise de requisitos: A extração dos requisitos de um desejado produto de software é a primeira tarefa na sua criação. Embora o cliente, provavelmente, acredite saber o que o software deva fazer, esta tarefa requer habilidade e experiência em engenharia de software para reconhecer a incompletude, ambigüidade ou contradição nos requisitos.

Especificação: A especificação é a tarefa de descrever precisamente o software que será escrito, preferencialmente de uma forma matematicamente rigorosa. Na prática, somente especificações mais bem sucedidas foram escritas para aplicações bem compreendidas e afinadas que já estavam bem desenvolvidas, embora sistemas de software de missão crítica sejam freqüentemente bem especificados antes do desenvolvimento da aplicação.

Arquitetura de Software: A arquitetura de um sistema de software remete a uma representação abstrata daquele sistema. Arquitetura é concernente à garantia de que o sistema de software irá ao encontro de requisitos do produto, como também assegurar que futuros requisitos possam ser atendidos. A etapa da arquitetura também direciona as interfaces entre os sistemas de software e outros produtos de software, como também com o hardware básico ou com o sistema operacional.

Implementação (ou codificação): A transformação de um projeto para um código

deve ser a parte mais evidente do trabalho da engenharia de software, mas não necessariamente a sua maior porção.

Teste: Teste de partes do software, especialmente onde tenha sido codificado por dois ou mais engenheiros trabalhando juntos, é um papel da engenharia de software.

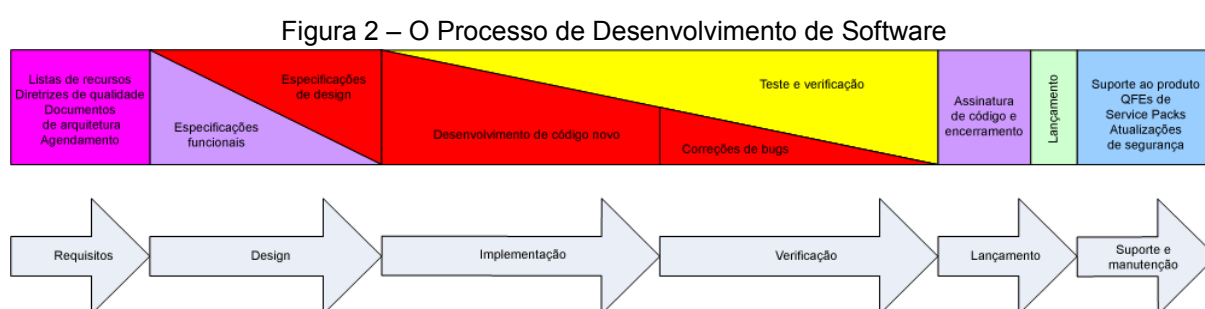
Documentação: Uma importante tarefa é a documentação do projeto interno do software para propósitos de futuras manutenções e aprimoramentos.

Suporte e Treinamento de Software: Uma grande porcentagem dos projetos de software falham pelo fato de o desenvolvedor não perceber que não importa quanto tempo a equipe de planejamento e desenvolvimento irá gastar na criação do software se ninguém da organização irá usá-lo. As pessoas ocasionalmente resistem à mudança e evitam aventurar-se em áreas pouco familiares. Então, como parte da fase de desenvolvimento, é muito importante o treinamento para os usuários de software mais entusiasmados, alternando o treinamento entre usuários neutros e usuários favoráveis ao software. Usuários irão ter muitas questões e problemas de software os quais conduzirão para a próxima fase.

Manutenção: A manutenção e melhoria de software lidam com a descoberta de novos problemas e requisitos. Ela pode tomar mais tempo que o gasto no desenvolvimento inicial do mesmo. Não somente pode ser necessário adicionar códigos que combinem com o projeto original, mas determinar como o software trabalhará em algum ponto depois da manutenção estar completa, pode requerer um significativo esforço por parte de um engenheiro de software. Cerca de $\frac{2}{3}$ de todos os engenheiros de software trabalham com a manutenção, mas estas estatísticas podem estar enganadas. Um pequena parte destes trabalha na correção de erros. A maioria das manutenções são para ampliar os sistemas para novas funcionalidades,

as quais, de diversas formas, podem ser consideradas um novo trabalho.

O processo de desenvolvimento de software tem sido objetivo de vários padrões, que visam a certificação de empresas como possuidoras de um processo de desenvolvimento, o que garantiria certo grau de confiança aos seus contratantes.



Há tempos, vem se tentando encontrar um processo ou metodologia previsível e repetível que melhore a produtividade e qualidade. Alguns tentaram sintetizar e formalizar a tarefa aparentemente incontrolável de escrever um software. Outros aplicaram técnicas de gerenciamento de projeto na escrita de software. Sem o gerenciamento de projeto, projetos de software podem facilmente sofrer atraso ou estourar o orçamento. Como um grande número de projetos de software não atendem suas expectativas em termos de funcionalidades, custo, ou cronograma de entrega, ainda não existe um modelo de processo perfeito para todas aplicações.

O mais antigo e bem conhecido processo é o **Modelo em cascata**, onde os desenvolvedores seguem estes passos em ordem. Eles estabelecem os requisitos, os analisam, projetam uma abordagem para solução, arquitetam um esboço do software, implementam o código, testam (inicialmente os testes unitários e então os testes de sistema), implantam e mantêm. Depois que cada passo é terminado, o processo segue para o próximo passo, tal como construtores que não revisam as fundações de uma casa depois que as paredes foram erguidas (PRESSMAN, 2006). Se as iterações não são incluídas no planejamento, o processo não tem meios para

corrigir os erros nas etapas iniciais (por exemplo, nos requisitos), então o processo inteiro da engenharia de software deve ser executado até o fim, resultando em funcionalidades de software desnecessárias ou sem uso. Para isso, deve-se fazer o implemento dos requisitos anteriormente analisados pelo programador.

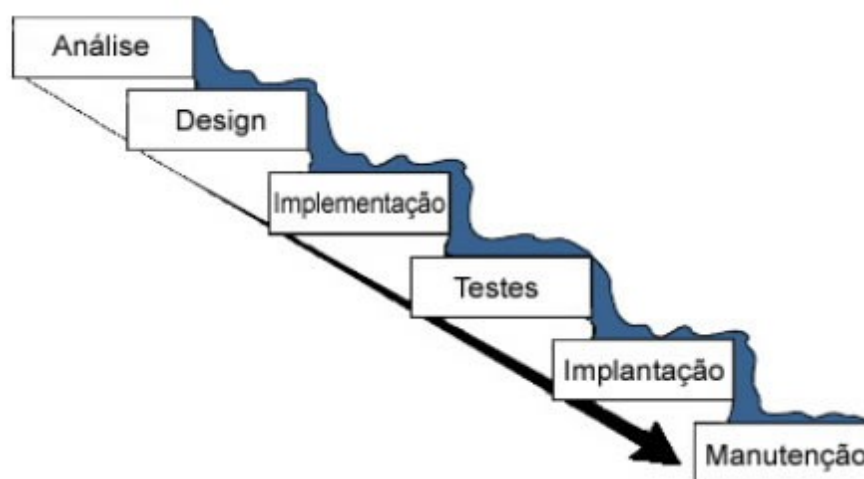


Figura 3: Modelo em Cascata

O **Desenvolvimento iterativo e incremental** prescreve a construção de uma porção pequena, mas abrangente, do projeto de software para ajudar a todos os envolvidos a descobrir cedo os problemas ou suposições, falhas que possam levar ao desastre (PRESSMAN, 2006). O processo iterativo é preferido por desenvolvedores porque lhes fornece um potencial para atingir os objetivos de projeto de um cliente que não sabe exatamente o que quer, ou quando não se conhece bem todos os aspectos da solução.

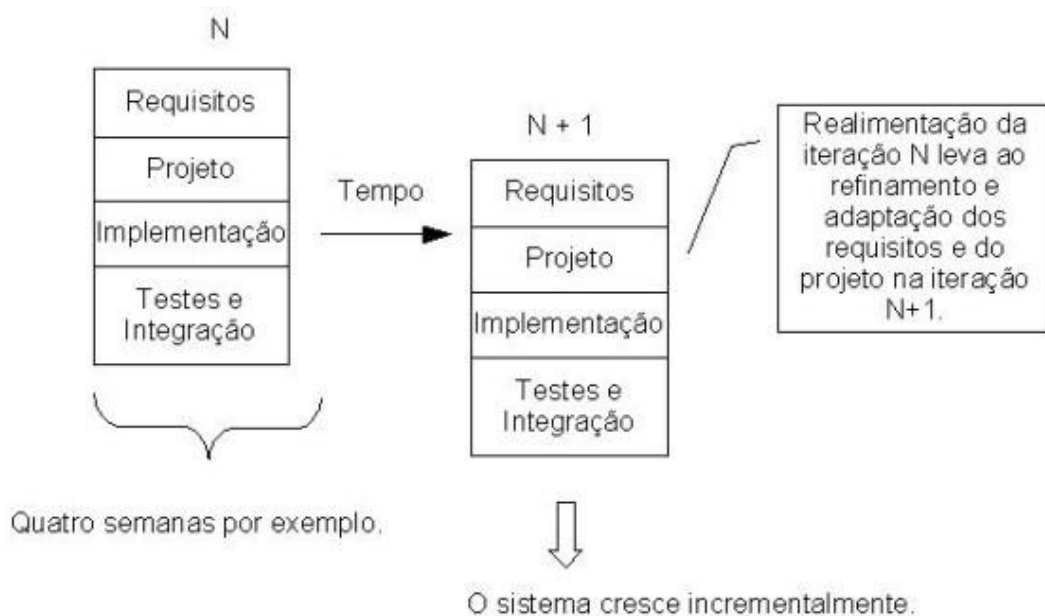


Figura 4 – Desenvolvimento Iterativo e Incremental

Processos de desenvolvimento ágil de software são construídos com os fundamentos do desenvolvimento iterativo. Os processos ágeis usam o feedback, mais que o planejamento, como seus mecanismos de controle primário. O feedback é produzido por testes regulares e das versões do software desenvolvido.

Os processos em Desenvolvimento ágil de software parecem ser mais eficientes do que as metodologias antigas. Utiliza menos tempo do programador no desenvolvimento de softwares funcionais de alta qualidade, mas tem a desvantagem de ter uma perspectiva de negócio que não provê uma capacidade de planejamento em longo prazo. Em essência, eles provem mais funcionalidades por custo/benefício, mas não dizem exatamente o que a funcionalidade irá fazer.

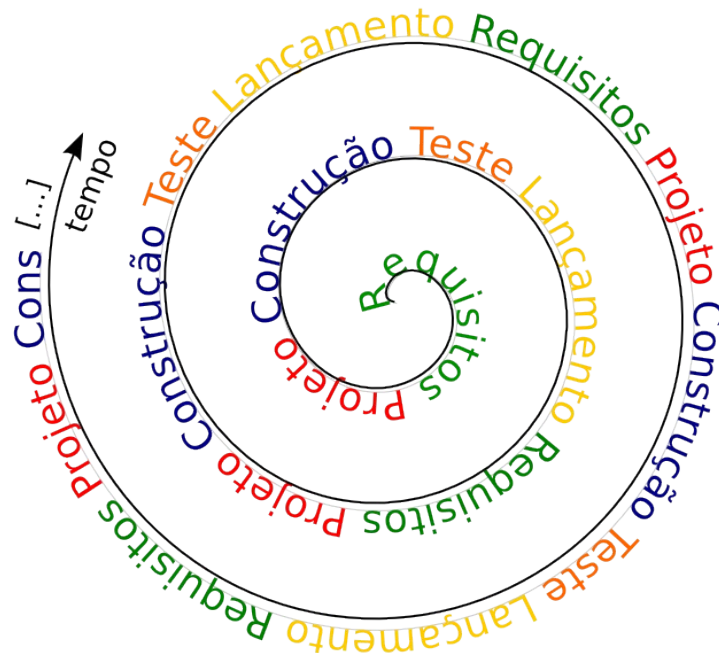


Figura 5 – Desenvolvimento ágil

Existem várias metodologias que podem ser consideradas como abordagens ágeis, entre elas: Scrum, Programação extrema, FDD, Crystal Clear, DSDM entre outras.

A Programação Extrema (XP), é o mais bem conhecido processo ágil. Em XP, as fases são continuadas em passos extremamente pequenos (ou contínuos) comparados aos processos antigos. A primeira passada (iteração incompleta) através das etapas deve levar um dia ou uma semana, ao invés de meses ou anos para cada fase completa do modelo em cascata. Primeiramente, escreve-se um autômato de teste, que provê objetivos concretos para o desenvolvimento. Depois é codificado (por um par de programadores), este passo está completo quando todos os testes se concluem, e os programadores não pensam em nada mais que possa ser testado. Projetistas e Arquitetos executam uma refatoração do código. O projeto é feito por pessoas que estão codificando. O sistema incompleto mas funcional é

O RUP foi usado como base para esse projeto, de maneira simplificada, por se tratar de uma metodologia estável, amplamente difundida e que reúne muitas vantagens dos processos de desenvolvimento expostos anteriormente (MARTINS, 2005).

O RUP usa a abordagem da orientação a objetos em sua concepção e é projetado e documentado utilizando a notação UML (Unified Modeling Language) para ilustrar os processos em ação. Utiliza técnicas e práticas aprovadas comercialmente.

É um processo considerado pesado e preferencialmente aplicável a grandes equipes de desenvolvimento e a grandes projetos, porém o fato de ser amplamente customizável torna possível que seja adaptado para projetos de qualquer escala. Para a gerência do projeto, o RUP provê uma solução disciplinada de como assinalar tarefas e responsabilidades dentro de uma organização de desenvolvimento de software.

O RUP é, por si só, um produto de software. É modular e automatizado, e toda a sua metodologia é apoiada por diversas ferramentas de desenvolvimento integradas e vendidas pela IBM através de seus "Rational Suites".

O RUP define as seguintes linhas-mestras e esqueletos (templates) para os membros da equipe de um ciclo de produção: parte do cliente, e uma avaliação do progresso do projeto pela sua gerência. Além disso, ajuda os programadores a manterem-se concentrados no projeto.

Uma documentação apropriada é essencial para qualquer grande projeto; note-se que o RUP descreve como documentar a funcionalidade, restrições de sistema, restrições de projeto e requisitos de negócio.

Os casos de uso e os cenários são exemplos de artefatos dependentes do processo,

que têm sido considerados muito mais eficazes na captura de requisitos funcionais.

A arquitetura baseada em componentes cria um sistema que pode ser facilmente extensível, promovendo a reutilização de software e um entendimento intuitivo. Um componente normalmente se relaciona com um objeto na programação orientada a objetos.

O RUP oferece uma forma sistemática para construir este tipo de sistema, focando-se em produzir uma arquitetura executável nas fases iniciais do projeto, ou seja, antes de comprometer recursos em larga escala.

Ao abstrair a programação do seu código e representá-la utilizando blocos de construção gráfica, o RUP consegue uma maneira efetiva de se ter uma visão geral de uma solução. O uso de modelos visuais também pode permitir que indivíduos de perfil menos técnico (como clientes) tenham um melhor entendimento de um dado problema, e assim se envolvam mais no projeto como um todo.

A linguagem de modelagem UML tornou-se um padrão industrial para representar projetos, e é amplamente utilizada pelo RUP.

Não assegurar a qualidade do software é a falha mais comum em todos os projetos de sistemas computacionais. Normalmente pensa-se em qualidade de software após o término dos projetos, ou a qualidade é responsabilidade de uma equipe diferente da equipe de desenvolvimento. O RUP visa auxiliar no controle do planejamento da qualidade, verificando-a na construção de todo o processo e envolvendo todos os membros da equipe de desenvolvimento.

Em todos os projetos de software a existência de mudanças é inevitável. O RUP define métodos para controlar e monitorar mudanças. Como uma pequena mudança

pode afetar aplicações de formas inteiramente imprevisíveis, o controle de mudanças é essencial para o sucesso de um projeto.

O RUP também define áreas de trabalho seguras, garantindo a um programador que as mudanças efetuadas noutro sistema não afetarão o seu sistema.

Até agora estas linhas de guia são gerais, a serem aderidas ao percorrer do ciclo de vida de um projeto. As fases indicam a ênfase que é dada no projeto em um dado instante. Para capturar a dimensão do tempo de um projeto, o RUP divide o projeto em quatro fases diferentes:

- **Concepção:** ênfase no escopo do sistema
- **Elaboração:** ênfase na arquitetura
- **Construção:** ênfase no desenvolvimento
- **Transição:** ênfase na implantação

O RUP também se baseia nos 4 P's:

- Pessoas
- Projeto
- Produto
- Processo

As fases são compostas de iterações. As iterações são janelas de tempo; as iterações possuem prazo definido enquanto as fases são objetivas.

Todas as fases geram artefatos. Estes serão utilizados nas próximas fases e

documentam o projeto, além de permitir melhor acompanhamento. Segue um modelo de aplicação do RUP, usado neste projeto.

	Requisitos	Análise	Arquitetura	Projeto	Codificação	Testes	Implantação	kit/marco
Concepção	<ul style="list-style-type: none"> - Documento de Visão - Requisitos Funcionais - Requisitos Funcionais Detalhados - Lista dos casos de uso - Diagramas de Casos de Uso - Proposta técnica (acordo) 	<ul style="list-style-type: none"> - Definição dos Pacotes do Sistema - Diagrama de Atividades (Glossário) - Especificação Suplementar 	<ul style="list-style-type: none"> - Identificação de Requisito Não Funcionais 			<ul style="list-style-type: none"> - Plano de Testes e de Qualidade de SW 	<ul style="list-style-type: none"> - Preparação do Ambiente de Análise e Projeto 	<ul style="list-style-type: none"> - Kit de Requisitos - Proposta Técnica
Elaboração	<ul style="list-style-type: none"> - Descrição dos casos de uso - Requisição de Mudanças - Planilha de Análise por pontos de Função - Análise de Risco 	<ul style="list-style-type: none"> - Diagrama de Classes de Negócio - Contrato do sistema - Diagrama de Estados - Diagrama de Entidade-Relacionamento 	<ul style="list-style-type: none"> - Descrição da Arquitetura de SW - Protótipo Arquitetural - Diagramas de Implantação 	<ul style="list-style-type: none"> - Diagrama de Classes de Projeto - Projeto Físico de Dados - Diagramas de Sequência 	<ul style="list-style-type: none"> - Protótipo Navegacional 	<ul style="list-style-type: none"> - Roteiro de Testes 	<ul style="list-style-type: none"> - Preparação do Ambiente de Desenvolvimento 	<ul style="list-style-type: none"> - Kit do Desenvolvedor - Domínio Conceitual
Construção	<ul style="list-style-type: none"> - Revisão nos Documentos de Requisitos 	<ul style="list-style-type: none"> - Revisões nos Documentos de Análise 	<ul style="list-style-type: none"> - Revisões nos Docs de Arquitetura 	<ul style="list-style-type: none"> - Revisões nos Documentos de Projeto 	<ul style="list-style-type: none"> - Codificação de Componentes - Testes Unitários 	<ul style="list-style-type: none"> - Revisão do Código - Testes de Integração - Testes de Sistema 	<ul style="list-style-type: none"> - Preparação do Ambiente de Homologação 	<ul style="list-style-type: none"> - Versão Alfa - Implementação do Produto
Transição				<ul style="list-style-type: none"> - Revisões nos Docs de Projeto 	<ul style="list-style-type: none"> - Aplicação de Correções em Componentes - Testes Unitários 	<ul style="list-style-type: none"> - Homologação da versão Beta 	<ul style="list-style-type: none"> - Migração para o Ambiente de Produção 	<ul style="list-style-type: none"> - Produto / Sistema - Transição para Produção

Quadro 2- Modelo de aplicação do RUP

A fase de concepção contém os workflows necessários que as partes interessadas (stakeholders) concordem com os objetivos, arquitetura, e o planejamento do projeto. Se as partes interessadas tiverem bons conhecimentos, pouca análise será requerida então. Se não tiverem o conhecimento necessário, mais análise será requerida. Nesta etapa foram gerados o objetivo do sistema, especificado na introdução e o documento de visão, mostrados abaixo:

O setor de RH (Recursos Humanos) cadastra empresas prestadoras de

cursos, informando CGC, nome e endereço.
Setor de RH cadastra a verba liberada para os cursos por ano, origem e órgão.
Servidor do RH cadastra curso informando o código do curso na empresa, o código do curso no órgão, o nome, a empresa prestadora e o total de horas. Cadastra também os dados das possíveis turmas para o curso, com local, datas de início e fim e periodicidade das aulas.
O servidor interessa-se por um determinado curso, inscrevendo-se no mesmo.
O chefe do servidor confirma o interesse do servidor no curso. O status da inscrição vai para “autorizado”. O sistema verifica se não há sobreposição de cursos.
O setor de RH informa que há verba para o curso e, estando o curso autorizado pelo chefe, informa isso no sistema. O status da inscrição vai para “existe verba”.
Estando o responsável pelo órgão de acordo, este viabiliza o curso. O status do mesmo vai para “verba alocada”. A verba cadastrada pelo RH sofre decréscimo para o pagamento do curso, ficando a parte pertinente como “alocada” e o restante “disponível”.
Cria-se um processo administrativo para a autorização final da inscrição. Após a aprovação deste PA, o status da inscrição vai para “inscrição confirmada”. O sistema envia um email para o servidor informando que ele está inscrito no curso, com todos os dados pertinentes.
Setor de RH emite uma lista de presença para o curso, com cada dia e alunos matriculados.
Após o término do curso, servidor cadastra a sua avaliação, de acordo com o modelo do RH.
Setor de RH homologa a avaliação e cadastra o aproveitamento do aluno (Aprovado, Desistente com Motivação, Desistente sem Motivação ou Reprovado)
RH informa se o aluno concluiu ou não o curso, com justificativa caso não

tenha concluído. O status da inscrição é colocado como “concluído” ou “cancelado”.
Setor de RH cadastra o certificado digitalizado do curso, a nota de empenho, a nota de crédito e a ordem de pagamento. O status da inscrição vai para “finalizada”. A verba que estava “alocada” vai para “gasta”.
Setor de RH não homologa o curso. O status do mesmo vai para “não homologado”. O valor “alocado” retorna para a verba “disponível”.
Setor de RH gera estatística da avaliação do curso.
O Setor de RH efetua os seguintes cadastros de apoio: Servidores, Servidores externos, Instrutores, Cursos/Eventos, Turmas (agrupamento de inscrições).
Setor de RH consulta os servidores com mais de x horas de cursos em um intervalo de tempo.
Setor de RH consulta o custo de treinamento por hora/aula e aluno, treinados, média de treinamentos realizados por servidor, “finalizados” e “não homologados”.

Quadro 3 – Documento de Visão

Nesta etapa também foram gerados o diagrama de casos de uso e o diagrama de atividades, que serão mostrados no anexo I.

Como cita o RUP, o ideal é que sejam feitas iterações. Porém estas devem ser bem definidas quanto a sua quantidade e objetivos.

A **fase de elaboração** será apenas para o projeto do sistema, buscando complementar o levantamento / documentação dos casos de uso, voltado para a arquitetura do sistema, revisa a modelagem do negócio para os projetos e inicia a versão do manual do usuário. Deve-se aceitar: Visão geral do produto (incremento e integração) está estável? O plano do projeto é confiável? Custos são admissíveis? Nesta etapa foram gerados outros diagramas da UML, mostrados no anexo I.

Na **fase de construção**, começa o desenvolvimento físico do software, produção de códigos, testes alfa e beta. Deve-se aceitar testes, e processos de testes estáveis, e se os códigos do sistema constituem "baseline".

Na **fase de transição** ocorre a entrega do software, é realizado o plano de implantação e entrega, acompanhamento e qualidade do software. Produtos (versões) devem ser entregues, e ocorrer a satisfação do cliente.

O RUP é baseado em um conjunto de princípios de desenvolvimento de software e melhores práticas, por exemplo:

- Desenvolvimento de software iterativo
- Gerenciamento de requisitos
- Uso de arquitetura baseada em componente
- Modelagem visual de software
- Verificação da qualidade do software
- Controle de alteração no software

O RUP usa desenvolvimento iterativo e incremental pelas seguintes razões:

- 1) A integração é feita passo a passo durante o processo de desenvolvimento, limitando-se cada passo a poucos elementos.
- 2) A integração é menos complexa, reduzindo seu custo e aumentando sua eficiência.
- 3) Partes separadas de projeto e/ou implementação podem ser facilmente identificadas para posterior reuso.

- 4) Mudanças de requisitos são registradas e podem ser acomodadas.
- 5) Os riscos são abordados no início do desenvolvimento e cada iteração permite a verificação de riscos já percebidos bem como a identificação de novos.
- 6) Arquitetura de software é melhorada através de um escrutínio repetitivo.

Usando iterações, um projeto terá um plano geral, como também múltiplos planos de iteração. O envolvimento dos stakeholders é freqüentemente encorajado a cada entrega. Desta maneira, as entregas servem como uma forma de se obter o comprometimento dos envolvidos, promovendo também uma constante comparação contra os requisitos e a prontidão da organização para as pendências que surgem.

O Gerenciamento de requisitos no RUP está concentrado em encontrar as necessidades do usuário final pela identificação e especificação do que ele necessita e identificando aquilo que deve ser mudado. Isto traz como benefícios:

A correção dos requisitos gera a correção do produto, as necessidades dos usuários são encontradas.

As características necessárias serão incluídas, reduzindo o custo de desenvolvimentos posteriores.

RUP sugere que o gerenciamento de requisitos tem que seguir estas atividades:

- 1) Análise do problema é concordar com o problema e criar medições que irão provar seu valor para a organização.
- 2) Entender as necessidades de seus stakeholders é compartilhar o problema e valores com os stakeholders-chave e levantar quais as necessidades que estão envolvidas na elaboração da idéia.
- 3) Definir o problema é a definição das características das necessidades e

esquematizar os casos de uso, atividades que irão facilmente mostrar os requerimentos de alto-nível e esboçar o modelo de uso do sistema.

Gerenciar o escopo do sistema trata das modificações de escopo que irão ser comunicadas baseadas nos resultados do andamento e selecionadas na ordem na qual os fluxogramas de casos de uso são atacados.

Refinar as definições do sistema trata do detalhamento dos fluxogramas de caso de uso com os stakeholders de forma a criar uma especificação de requerimentos de software que pode servir como um contrato entre o seu grupo e o do cliente e poderá guiar as atividades de teste e projeto.

Gerenciamento das mudanças de requisitos trata de como identificar as chegadas das mudanças de requerimento num projeto que já começou.

Arquitetura baseada em componentes cria um sistema que é facilmente extensível, intuitivo e de fácil compreensão e promove a reusabilidade de software. Um componente freqüentemente se relaciona com um conjunto de objetos na programação orientada ao objeto.

Arquitetura de software aumenta de importância quando um sistema se torna maior e mais complexo. RUP foca na produção da arquitetura básica nas primeiras iterações. Esta arquitetura então se torna um protótipo nos ciclos iniciais de desenvolvimento. A arquitetura desenvolve-se em cada iteração para se tornar a arquitetura final do sistema. RUP também propõem regras de projeto e restrições para capturar regras de arquitetura. Pelo desenvolvimento iterativo é possível identificar gradualmente componentes os quais podem então ser desenvolvidos, comprados ou reusados. Estes componentes são freqüentemente construídos em infra-estruturas existentes tais como CORBA e COM, ou JavaEE, ou PHP

Abstraindo sua programação do seu código e representando-a usando blocos de construção gráfica constitui-se de uma forma efetiva de obter uma visão geral de uma solução. Usando esta representação, recursos técnicos podem determinar a melhor forma para implementar a dado conjunto de interdependências lógicas. Isto também constrói uma camada intermediária entre o processo de negócio e o código necessário através da tecnologia da informação. Um modelo neste contexto é uma visualização e ao mesmo tempo uma simplificação de um projeto complexo. RUP especifica quais modelos são necessários e porque.

A Linguagem modelagem unificada (UML) pode ser usada para modelagem de Casos de Uso, diagrama de classes e outros objetos. RUP também discute outras formas para construir estes modelos.

Garantia da qualidade de software é o ponto mais comum de falha nos projetos de software, desde que isto é freqüentemente algo que não se pensa previamente e é algumas vezes tratado por equipes diferentes. O RUP ajuda no planejamento do controle da qualidade e cuida da sua construção em todo processo, envolvendo todos os membros da equipe. Nenhuma tarefa é especificamente direcionada para a qualidade; o RUP assume que cada membro da equipe é responsável pela qualidade durante todo o processo. O processo foca na descoberta do nível de qualidade esperado e provê testes nos processos para medir este nível.

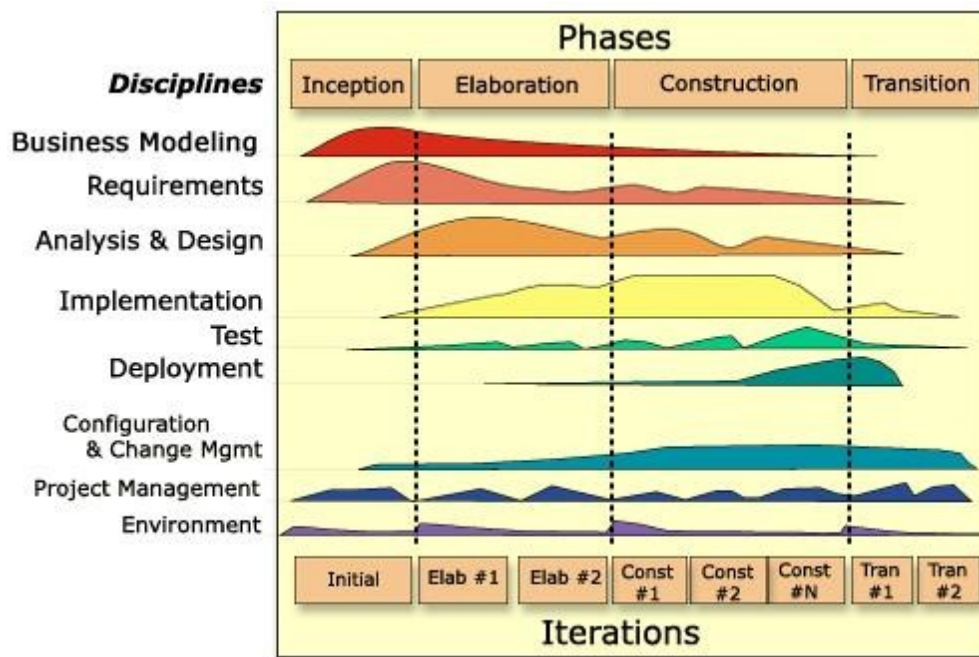


Figura 7 – O RUP.

Em todos os projetos de software, mudanças são inevitáveis. RUP define métodos para controlar, rastrear e monitorar estas mudanças. RUP também define espaços de trabalho seguros (do inglês *secure workspaces*), garantindo que um sistema de engenharia de software não será afetado por mudanças em outros sistemas. Este conceito é bem aderente com arquiteturas de software baseados em componentes

I.3 UML

A Unified Modeling Language (UML) é uma linguagem de modelagem não proprietária de terceira geração. Segundo Larman (2000), a UML não é uma metodologia de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como projetar seu sistema, mas ela lhe auxilia a

visualizar seu desenho e a comunicação entre objetos. Foi utilizada amplamente neste projeto, por se tratar de um padrão para a análise de sistemas e por estar intimamente ligada ao RUP.

Basicamente, a UML permite que desenvolvedores visualizem os produtos de seu trabalho em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significados, isto é, semântica. É uma notação independente de processos, embora o RUP (Rational Unified Process) tenha sido especificamente desenvolvido utilizando a UML.

É importante distinguir entre um modelo UML e um diagrama (ou conjunto de diagramas) de UML. O último é uma representação gráfica da informação do primeiro, mas o primeiro pode existir independentemente. O XMI (XML Metadata Interchange) na sua versão corrente disponibiliza troca de modelos mas não de diagramas.

Embora a UML defina uma linguagem precisa, ela não é uma barreira para futuros aperfeiçoamentos nos conceitos de modelagem. O desenvolvimento da UML foi baseado em técnicas antigas e marcantes da orientação a objetos, mas muitas outras influenciarão a linguagem em suas próximas versões. Muitas técnicas avançadas de modelagem podem ser definidas usando UML como base, podendo ser estendida sem se fazer necessário redefinir a sua estrutura interna.

A UML será a base para muitas ferramentas de desenvolvimento, incluindo modelagem visual, simulações e ambientes de desenvolvimento. Em breve, ferramentas de integração e padrões de implementação baseados em UML estarão disponíveis para qualquer um.

A UML tem origem na compilação das "melhores práticas de engenharia" que

provaram ter sucesso na modelagem de sistemas grandes e complexos. Sucedeu aos conceitos de Booch, OMT (Rumbaugh) e OOSE (Jacobson) fundindo-os numa única linguagem de modelagem comum e largamente utilizada. A UML pretende ser a linguagem de modelagem padrão para modelar sistemas concorrentes e distribuídos (LARMAN, 2000).

Abaixo, segue uma figura com os principais diagramas:

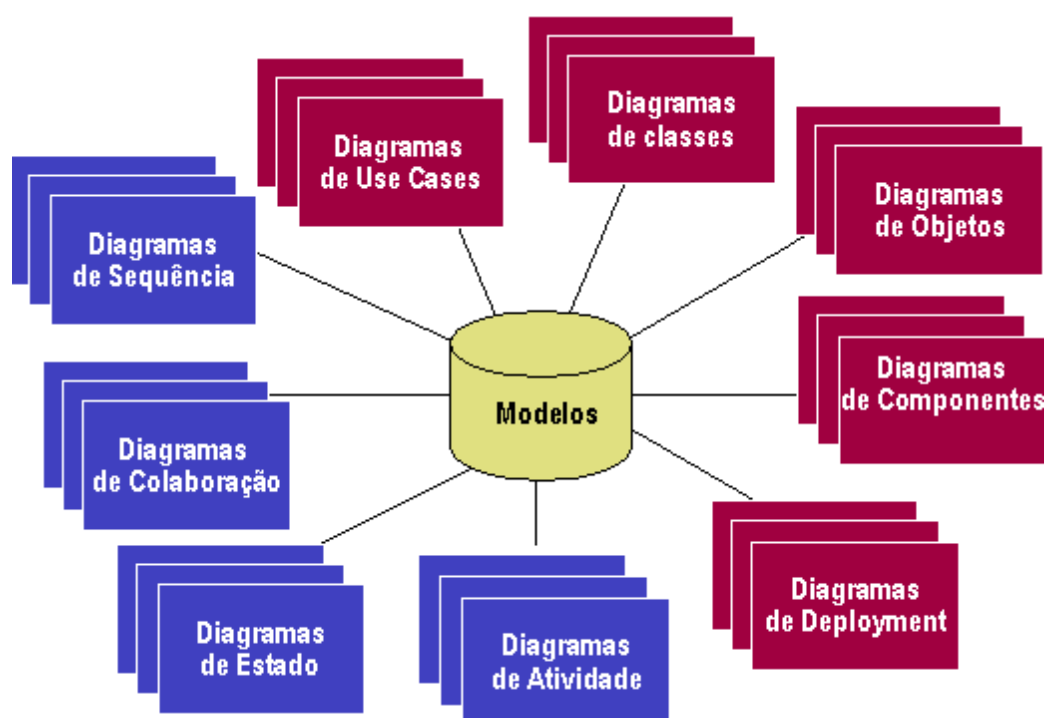


Figura 8 – Diagramas da UML

A UML integrou muitas idéias adversas, e esta integração acelera o uso do desenvolvimento de softwares orientados a objetos.

I.4 JAVA

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um "bytecode". Para a execução dos programas, o sistema operacional onde o programa for executado deve possuir uma JVM (Java Virtual Machine). Este método de compilação e interpretação permite grande portabilidade, pois um código compilado em uma máquina com sistema operacional Windows pode ser executado em uma máquina rodando Linux, bastando que ambas possuam sua devida máquina virtual instalada. Muitas pessoas acreditam que por causa desse processo, o código interpretado Java tem baixo desempenho, mas isso não é verdade. Durante muito tempo esta foi uma afirmação verdadeira. Porém novos avanços têm tornado o compilador dinâmico (a JVM), em muitos casos, mais eficiente que o compilador estático (HAGGAR, 2003).

Outra grande vantagem do Java é a sua total orientação a objetos, possibilitando nativamente a abordagem deste paradigma. Sua forte tipagem também assegura o tratamento de variáveis.

Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Em 2004 Java atingiu a marca de 3 milhões de desenvolvedores em todo mundo. Java continuou crescendo e hoje é uma referência no mercado de desenvolvimento de software. Java tornou-se popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em web browsers, mainframes, SOs, celulares, palmtops e cartões inteligentes, entre outros.

A linguagem Java foi projetada tendo em vista os seguintes objetivos:

- **Orientação a objeto** - Baseado no modelo de Smalltalk e Simula67;
- **Portabilidade** - Independência de plataforma - "write once run anywhere";
- **Recursos de Rede** - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP;
- **Segurança** - Pode executar programas via rede com restrições de execução;

Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:

- Sintaxe similar a Linguagem C/C++ e principalmente, a C#.
- Facilidades de Internacionalização - Suporta nativamente caracteres Unicode;
- Simplicidade na especificação, tanto da linguagem como do "ambiente" de execução (JVM);
- É distribuída com um vasto conjunto de bibliotecas (ou APIs);
- Possui facilidades para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa);
- Desalocação de memória automática por processo de coletor de lixo (garbage collector);
- Carga Dinâmica de Código - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

Java hoje já possui um desempenho próximo do C++. Isto é possível graças a otimizações como a compilação especulativa, que aproveita o tempo ocioso do processador para pré-compilar bytecode para código nativo. Outros mecanismos ainda mais elaborados como o HotSpot da Sun, que guarda informações disponíveis somente em tempo de execução (ex.: número de usuários, processamento usado, memória disponível), para otimizar o funcionamento da JVM, possibilitando que a

JVM vá "aprendendo" e melhorando seu desempenho. Isto é uma realidade tão presente que hoje é fácil encontrar programas corporativos e de missão crítica usando tecnologia Java. No Brasil, por exemplo, a maioria dos Bancos utiliza a tecnologia Java para construir seus home banks, que são acessados por milhares de usuários diariamente. Grandes sítios como o eBay utilizam Java para garantir alto desempenho.

Essa implementação no entanto tem algumas limitações intrínsecas. A pré-compilação exige tempo, o que faz com que programas Java demorem um tempo significativamente maior para começarem a funcionar. Soma-se a isso o tempo de carregamento da máquina virtual. Isso não é um grande problema para programas que rodam em servidores e que deveriam ser inicializados apenas uma vez. No entanto isso pode ser bastante indesejável para computadores pessoais onde o usuário deseja que o programa rode logo depois de abri-lo. A próxima versão da máquina virtual produzida pela Sun promete novos recursos que irão minimizar este fato.

O Java ainda possui uma outra desvantagem considerável em programas que usam bastante processamento numérico. O padrão Java tem uma especificação rígida de como devem funcionar os tipos numéricos. Essa especificação não condiz com a implementação de pontos flutuantes na maioria dos processadores o que faz com que o Java seja significativamente mais lento para estas aplicações quando comparado a outras linguagens.

Os bytecodes produzidos pelos compiladores Java podem ser usados num processo de engenharia reversa para a recuperação do programa-fonte original. Esta é uma característica que atinge em menor grau todas as linguagens compiladas. No entanto já existem hoje tecnologias que "embaralham" e até mesmo criptografam os bytecodes praticamente impedindo a engenharia reversa (DEITEL, 2003).

É possível desenvolver aplicações em Java através de vários ambientes de desenvolvimento integrado (IDE's). Dentre as opções mais utilizadas pode-se destacar:

Eclipse — um projeto aberto iniciado pela IBM;

NetBeans — um ambiente criado pela empresa Sun Microsystems;

JBuilder — um ambiente desenvolvido pela empresa Borland;

JDeveloper — uma IDE desenvolvida pela empresa Oracle;

JCreator — um ambiente desenvolvido pela Xinox;

BlueJ — um ambiente desenvolvido por uma faculdade australiana (considerado muito bom para iniciantes);

IntelliJ IDEA — uma IDE desenvolvida pela JetBrains (considerada por muitos a melhor IDE do mercado).

Devido as vantagens mencionadas e por ser gratuita, a linguagem Java foi escolhida para a implementação do projeto.

I.5 BANCO DE DADOS RELACIONAIS

Um Banco de Dados Relacional é um banco de dados que segue o Modelo Relacional. Apesar de existirem outros modelos; como o orientado a objetos, lista invertida e hierárquico; o modelo relacional é o mais difundido para sistemas comerciais, tendo sido escolhido para este projeto inclusive por critérios de conhecimento do autor sobre o assunto.

De forma mais detalhada, um Banco de Dados Relacional é um conceito abstrato que define maneiras de armazenar, manipular e recuperar dados estruturados unicamente na forma de tabelas, construindo um banco de dados (DATE, 1994).

O termo também é aplicável aos próprios dados, quando organizados dessa forma, ou a um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) – do inglês *relational database management system* (RDBMS) – um programa de computador que implementa a abstração.

Os Bancos de dados relacionais (BDR) surgiram em meados da década de 1970. Porém, apenas alguns anos mais tarde as empresas passaram a utilizá-los no lugar de arquivos planos (do inglês flat file), bancos de dados hierárquicos e em rede (Date, 1994).

Em 1985, Edgar Frank Codd, criador do modelo relacional, publicou um artigo onde definia 12 regras para que um Sistema Gerenciador de Banco de Dados (SGBD) fosse considerado relacional:

- **Regra Fundamental:** Um SGBD relacional deve gerenciar seus dados usando apenas suas capacidades relacionais
- **Regra da informação:** Toda informação deve ser representada de uma única forma, como dados em uma tabela
- **Regra da garantia de acesso:** Todo dado (valor atômico) pode ser acessado logicamente (e unicamente) usando o nome da tabela, o valor da chave primária da linha e o nome da coluna.
- **Tratamento sistemático de valores nulos:** Os valores nulos (diferente do zero, da string vazia, da string de caracteres em brancos e outros valores não nulos) existem para representar dados não existentes de forma sistemática e independente do tipo de dado.

- **Catálogo dinâmico on-line baseado no modelo relacional:** A descrição do banco de dados é representada no nível lógico como dados ordinários (isso é, em tabelas), permitindo que usuários autorizados apliquem as mesmas formas de manipular dados aplicada aos dados comuns ao consultá-las.
- **Regra da sub-linguagem compreensiva:** Um sistema relacional pode suportar várias linguagens e formas de uso, porém deve possuir ao menos uma linguagem com sintaxe bem definida e expressa por cadeia de caracteres e com habilidade de apoiar a definição de dados, a definição de visões, a manipulação de dados, as restrições de integridade, a autorização e a fronteira de transações.
- **Regra da atualização de visões:** Toda visão que for teoricamente atualizável será também atualizável pelo sistema.
- **Inserção, atualização e eliminação de alto nível:** A capacidade de manipular a relação base ou relações derivadas como um operador único não se aplica apenas a recuperação de dados, mas também a inserção, alteração e eliminação de dados.
- **Independência dos dados físicos:** Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas quaisquer que sejam as modificações na representação de armazenagem ou métodos de acesso internos.
- **Independência lógica de dados:** Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas quaisquer que sejam as mudanças de informação que permitam teoricamente a não alteração das tabelas base.
- **Independência de integridade:** As relações de integridade específicas de um banco de dados relacional devem ser definidas em uma sub-linguagem de dados e armazenadas no catálogo (e não em programas).
- **Independência de distribuição:** A linguagem de manipulação de dados deve possibilitar que as aplicações permaneçam inalteradas estejam os dados centralizados ou distribuídos fisicamente.
- **Regra da Não-subversão:** Se o sistema relacional possui uma linguagem de

baixo nível (um registro por vez), não deve ser possível subverter ou ignorar as regras de integridade e restrições definidas no alto nível (muitos registros por vez).

Os Bancos de Dados Relacionais foram desenvolvidos para prover acesso facilitado aos dados, possibilitando que os usuários utilizassem uma grande variedade de abordagens no tratamento das informações. Pois, enquanto em um banco de dados hierárquico os usuários precisam definir as questões de negócios de maneira específica, iniciando pela raiz do mesmo, nos Bancos de Dados Relacionais os usuários podem fazer perguntas relacionadas aos negócios através de vários pontos.

A linguagem padrão dos Bancos de Dados Relacionais é a Structured Query Language, ou simplesmente SQL, como é mais conhecida.

A arquitetura de um banco de dados relacional pode ser descrita de maneira informal ou formal. Na descrição informal estamos preocupados com aspectos práticos da utilização e usamos os termos tabela, linha e coluna. Na descrição formal estamos preocupados com a semântica formal do modelo e usamos termos como relação (tabela), tupla (linhas) e atributo (coluna).

Todos os dados de um banco de dados relacional (BDR) são armazenados em tabelas. Uma tabela é uma simples estrutura de linhas e colunas. Em uma tabela, cada linha contém um mesmo conjunto de colunas. Em um banco de dados podem existir uma ou centenas de tabelas, sendo que o limite pode ser imposto tanto pela ferramenta de software utilizada, quanto pelos recursos de hardware disponíveis no equipamento.

As tabelas associam-se entre si através de regras de relacionamentos, estas regras

consistem em associar um ou vários atributo de uma tabela com um ou vários atributos de outra tabela.

Exemplo: A tabela funcionário relaciona-se com a tabela cargo. Através deste relacionamento esta última tabela fornece a lista de cargos para a tabela funcionário.

Cada linha formada por uma lista ordenada de colunas representa um registro, ou tupla. Os registros não precisam conter informações em todas as colunas, podendo assumir valores nulos quando assim se fizer necessário.

Resumidamente, um registro é uma instância de uma tabela, ou entidade.

Exemplo: O empregado João é uma instância (registro) da tabela funcionário, e a função Analista Comercial é a instância (registro) da tabela cargo. Uma associação entre estas duas tabelas criaria a seguinte instância de relacionamento: Pedro é Analista Comercial, onde o verbo ser representa uma ligação entre os registros distintos.

As colunas de uma tabela são também chamadas de Atributos. Ao conjunto de valores que um atributo pode assumir chama-se domínio. Por exemplo: em um campo do tipo numérico, serão somente armazenados números.

O conceito mais similar a domínio é o de Tipo Abstrato de Dados em linguagens de programação, ou seja são meta-dados (dados acerca de dados).

As tabelas relacionam-se umas as outras através de chaves. Uma chave é um conjunto de um ou mais atributos que determinam a unicidade de cada registro.

Por exemplo, se um banco de dados tem como chaves Código do Produto e ID

Sistema, sempre que acontecer uma inserção de dados o sistema de gerenciamento de banco de dados irá fazer uma consulta para identificar se o registro já não se encontraria gravado na tabela. Neste caso, um novo registro não será criado, resultando esta operação apenas da alteração do registro existente.

A unicidade dos registros, determinada por sua chave, também é fundamental para a criação dos índices.

Temos dois tipos de chaves:

- Chave primária: (PK - Primary Key) é a chave que identifica cada registro dando-lhe unicidade. A chave primária nunca se repetirá.
- Chave Estrangeira: (FK - Foreign Key) é a chave formada através de um relacionamento com a chave primária de outra tabela. Define um relacionamento entre as tabelas e pode ocorrer repetidas vezes. Caso a chave primária seja composta na origem, a chave estrangeira também o será.

Com o advento do Modelo de Entidades e Relacionamentos foi causada uma confusão entre os termos relação e relacionamento

O Modelo Relacional, quando descrito de forma matemática, é definido como um modelo formado por relações (no sentido matemático) entre os domínios. Cada tupla é um elemento do conjunto relação.

Ou seja, a relação é a tabela.

Um relacionamento do Modelo de Entidades e Relacionamentos é uma associação entre entidades distintas. Não há relação direta entre o nome relacionamento e o nome relação. Porém, um relacionamento, do Modelo de Entidades e Relacionamentos é traduzido para a criação de atributos com chaves externas do Modelo Relacional. Esta tradução é feita ligando-se um campo de uma tabela X com

um campo de uma tabela Y, por meio da inclusão do campo chave da tabela Y como um campo (conhecido como chave estrangeira) da tabela X.

Por meio das chaves estrangeiras, é possível implementar restrições nos SGBDR.

Existem alguns tipos de relacionamentos possíveis no MER:

Um para um (1 para 1) - indica que as tabelas têm relação unívoca entre si. Você escolhe qual tabela vai receber a chave estrangeira;

Um para muitos (1 para N) - a chave primária da tabela que tem o lado 1 vai para a tabela do lado N. No lado N ela é chamada de chave estrangeira;

Muitos para muitos (N para N) - quando tabelas têm entre si relação n..n, é necessário criar uma nova tabela com as chaves primárias das tabelas envolvidas, ficando assim uma chave composta, ou seja, formada por diversos campos-chave de outras tabelas. A relação então se reduz para uma relação 1..n, sendo que o lado n ficará com a nova tabela criada.

Os relacionamento 1 para 1 e 1 para N podem ser mapeados diretamente em chaves estrangeiras nas tabelas originais. Já o relacionamento N para N exige o uso de uma tabela auxiliar. O uso do relacionamento 1 para 1 geralmente pode ser substituído por uma única tabela.

O modelo de entidade relacionamento é um dos artefatos principais no desenvolvimento de um software, uma vez que representa como os dados serão armazenados no banco de dados. Segue o M.E.R. elaborado para o sisTreina.

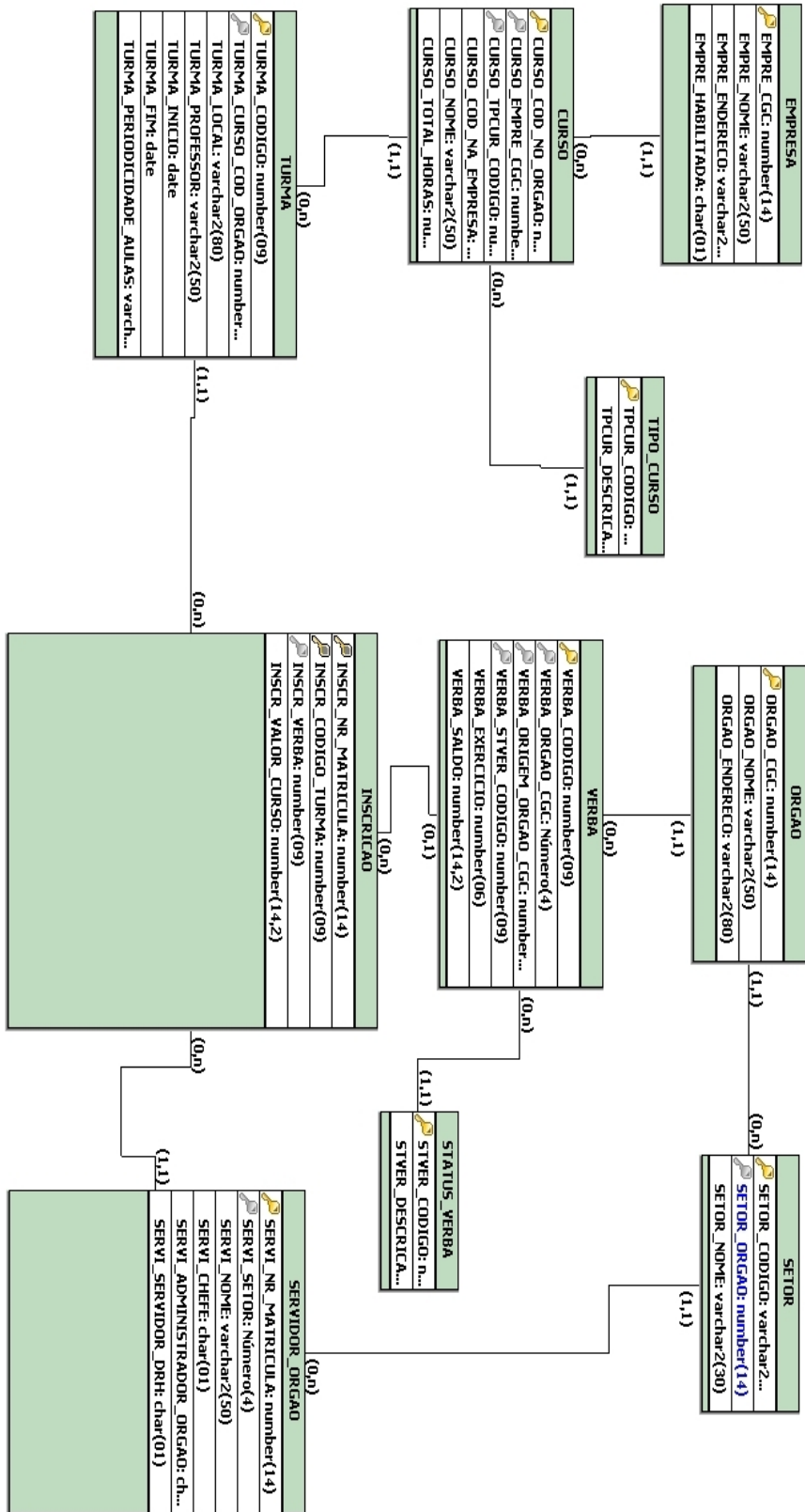


Figura 9 – Diagrama de Entidade Relacionamento – sisTreina

Normalização é o processo de organização eficiente dos dados dentro de um banco de dados cujos objetivos principais são:

- Eliminar dados redundantes (por exemplo, armazenando os mesmos dados em mais de uma tabela).
- Garantir que as dependências entre os dados façam sentido (armazenando apenas dados logicamente relacionados em uma tabela).

Existem cinco estágios de normalização, 1º, o 2º, o 3º, o 4º e o 5º. Para um banco de dados se encontrar em cada um desses estágios ou formas (denominadas formas normais), cada uma de suas tabelas deve atender a alguns pré-requisitos. Os pré-requisitos são cumulativos, isto é, para alcançar a 3ª forma normal (3NF), um banco de dados precisa atender aos pré-requisitos das 1ª e 2ª formas normais.

Os principais SGBDs relacionais existentes no mercado são: DB2, Ingres, InterBase, MySQL, Oracle, PostgreSQL, Progress, Microsoft SQL Server, Microsoft SQL Server, Express Edition, SQLite, Sybase, Informix, Firebird

I.6 MYSQL

O MySQL é um sistema de gerenciamento de banco de dados (SGBD) gratuito, que utiliza a linguagem SQL (Structured Query Language - Linguagem de Consulta Estruturada) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo, segundo o próprio site do produto (www.mysql.com, 2009)

Segundo ainda informações do site, entre os usuários do banco de dados MySQL estão: NASA, Friendster, Banco Bradesco, Dataprev, HP, Nokia, Sony, Lufthansa, U.S Army, US. Federal Reserve Bank, Associated Press, Alcatel, Slashdot, Cisco Systems e outros.

O MySQL foi criado na Suécia por dois suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius, que têm trabalhado juntos desde a década de 1980. Hoje seu desenvolvimento e manutenção empregam aproximadamente 70 profissionais no mundo inteiro, e mais de mil contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito dele.

O sucesso do MySQL deve-se em grande medida à fácil integração com o PHP incluído, quase que obrigatoriamente, nos pacotes de hospedagem de sites da Internet oferecidos atualmente. Empresas como Yahoo! Finance, MP3.com, Motorola, NASA, Silicon Graphics e Texas Instruments usam o MySQL em aplicações de missão crítica. A Wikipédia é um exemplo de utilização do MySQL em sites de grande audiência.

O MySQL hoje suporta Unicode, Full Text Indexes, replicação, Hot Backup, GIS, OLAP e muitos outros recursos.

Seguem algumas outras vantagens:

- Portabilidade (suporta praticamente qualquer plataforma atual);
- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, Python, Perl, PHP, ASP e Ruby)
- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de hardware;

- Facilidade de uso;
- É um Software Livre com base na GPL;
- Contempla a utilização de vários Storage Engines como MyISAM, InnoDB, Falcon, BDB, Archive, Federated, CSV, Solid...
- Suporta controle transacional;
- Suporta Triggers;
- Suporta Cursors (Non-Scrollable e Non-Updatable);
- Suporta Stored Procedures e Functions;
- Replicação facilmente configurável;
- Interfaces gráficas ([MySQL Toolkit]) de fácil utilização cedidos pela MySQL Inc.

Outra grande vantagem é ter código aberto e funcionar em um grande número de sistemas operacionais : Windows, Linux, FreeBSD, BSDI, Solaris, Mac OS X, SunOS, SGI, etc.

É reconhecido pelo seu desempenho e robustez e também por ser multi-tarefa e multi-usuário. A própria Wikipédia, usando o programa MediaWiki, utiliza o MySQL para gerenciar seu banco de dados, demonstrando que é possível utilizá-lo em sistemas de produção de alta exigência e em aplicações sofisticadas.

No passado, devido a não possuir (até a versão 3.x) funcionalidades consideradas essenciais em muitas áreas, como stored procedures, two-phase commit, subselects, foreign keys ou integridade referencial, era frequentemente considerado um sistema mais "leve" e para aplicações menos exigentes, sendo preterido por outros sistemas como o PostgreSQL.

O MySQL é somente o banco de dados, necessitando também de um software que interaja com o usuário a fim de guardar as informações (em um banco de dados)

que o usuário adicionou em um site da internet ou em um software.

O MySQL a partir da versão 4.1 adicionou suporte a Transações, SubSelects, Foreign Keys e Integridade Referencial. Esse suporte foi graças ao Storage Engine InnoDB, adquirido em 2005 pela Oracle.

Com a versão 5.0, o MySQL incorporou mais recursos avançados ao sistema, incluindo views , triggers, stored procedures, cursors e transações XA.

Devido às suas vantagens e pelo fato de ser gratuito, facilitando sua implantação em qualquer órgão público, o MySQL foi escolhido como SGBD oficial para o sisTreina, o que não impede o uso de outro SGBD com as devidas customizações.

I.7 ARQUITETURA WEB (TRES CAMADAS)

I.7.1 Arquitetura de uma camada

Os primeiros sistemas para o simultâneo de vários usuários eram constituídos por somente uma camada: um computador servidor, de considerável capacidade de processamento e armazenamento, hospedava o sistema. Terminais, conectados a este servidor, executavam a aplicação usando o hardware e o software dele. Esta arquitetura se aplica a ambientes controlados, ficando cada vez mais inaplicável a medida que os sistemas tinham que funcionar em locais geograficamente distantes.

I.7.2 Arquitetura de duas camadas

Com o tempo, e a difusão dos computadores pessoais, surgiu a arquitetura em duas camadas (cliente-servidor). Um ou mais computadores servidores disponibilizavam a aplicação e a base de dados. Os computadores clientes executavam a aplicação usando seu próprio ambiente de processamento, desonerando o servidor. Esta arquitetura ainda é usada até hoje, porém requer que as aplicações seja direcionadas para o ambiente dos computadores clientes

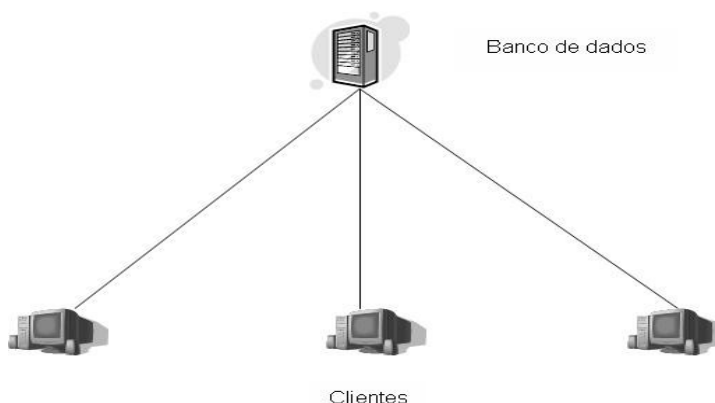


Figura 10 – Arquitetura em duas camadas

I.7.3 Arquitetura de três camadas

Com a advento da internet, os sistemas comerciais e corporativos também passaram a ser executados via arquitetura web ou arquitetura três camadas. A primeira camada é o computador cliente, o qual precisa possuir somente um programa chamado browser. A segunda camada é o servidor de aplicação, o qual recebe as requisições dos clientes e as processa, chamando, se necessário, a

terceira camada onde fica hospedado o banco de dados.

Esta arquitetura facilitou a difusão de sistemas em locais geograficamente distantes, uma vez que basta um computador possuir um browser e acesso a internet para pode acessar a aplicação. Além disso, tornou as aplicações mais portáteis, uma vez que elas não dependem do hardware e do sistema operacional que está tentando acessá-las. A camada de aplicação comunica-se somente com o browser.

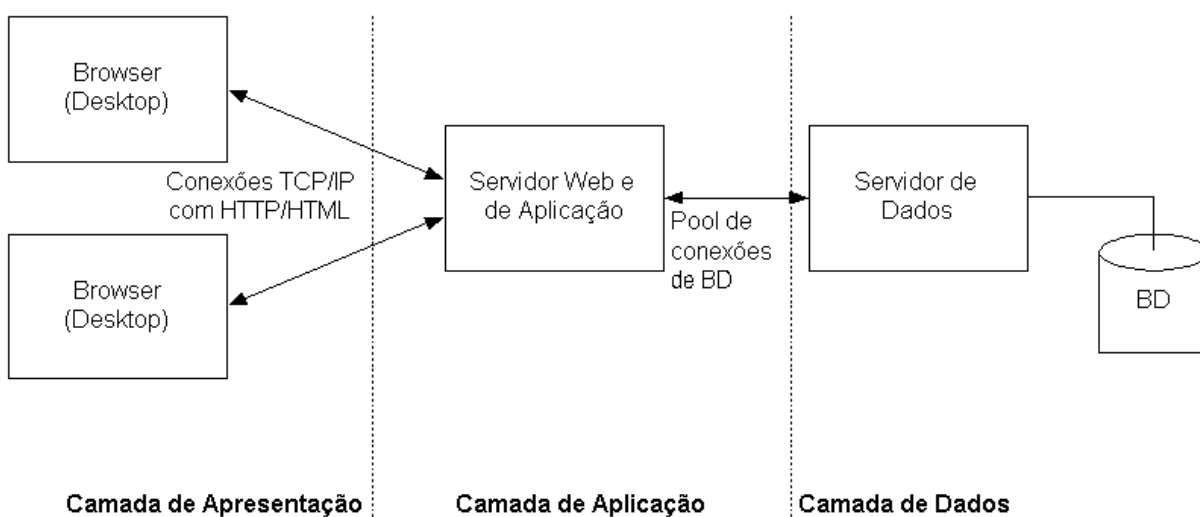


Figura 11- Arquitetura em três camadas

CAPÍTULO II - COMPARAÇÃO COM OUTROS SISTEMAS

Foi encontrado um sistema que efetua parte da proposta do sisTreina: o sistema Horus.

Os servidores do Ministério Público Federal precisam de um determinado número de horas de treinamento específico para poderem progredir em suas carreiras. O sistema Horus tem como finalidade cadastrar estes cursos e somar as respectivas horas. Ou seja, é um sistema de cadastro e consulta:

- Os servidores cadastram os treinamentos que não foram custeados pelo MPF
- O setor de recursos humanos cadastra os cursos custeados pelo MPF
- O setor de recursos humanos homologa o cadastramento
- Os servidores podem consultar os cursos já efetuados

Adicional de Qualificação

Formação

Registro de Treinamentos Treinamentos Particulares Treinamentos Institucionais Ajuda

ATENÇÃO: Aqui só deverão ser cadastrados os cursos não custeados pelo MPF!

Curso:	<input type="text"/>
Nível:	TREINAMENTO
Área de conhecimento:	Selecione uma área...
Data de início:	<input type="text"/> 1
Data de conclusão:	<input type="text"/> 1
Carga horária:	<input type="text"/>
Instituição de ensino (CNPJ):	<input type="text"/>
Instituição de ensino:	<input type="text"/>
Possui certificado / declaração ?	<input type="radio"/> Sim <input type="radio"/> Não

Salvar

Figura 12 – Tela de Cadastro de Cursos – sistema Horus

Treinamentos Particulares							
Treinamentos Institucionais							
Ajuda							
Curso	Área conhecimento	CH	Início	Fim	Instituição	Status	
1	Treinamento Em Gerência De Projetos Com M...	Gerenciamento E Administr...	20	25/08/2008	29/08/2008	Servigo Nacional De Aprendizagem Comercial-senac Arrj	Homologado
2	Java Security	Computação	24	19/10/2007	26/10/2007	Infnet - Instituto De Formação Internet	Pendente
3	Java Web Applications	Computação	48	12/09/2007	27/09/2007	Infnet - Instituto De Formação Internet	Pendente
4	Métodos De Revisão E Testes Em Projetos D...	Computação	42	20/08/2007	03/12/2007	Pontifícia Universidade Católica Do Rio De Janeiro	Pendente
5	Capacitação Em Análise De Pontos Por Função	Computação	16	23/09/2006	30/09/2006	Fatto - Consultoria E Sistemas Ltda	Pendente
6	I521 - Java Web Tier - Servlets E Jsp	Computação	40	23/09/2004	06/10/2004	Infnet - Instituto De Formação Internet	Pendente
7	Curso Aberto De Chefia E Liderança Empres...	Comércio E Administração(...	15	13/09/2004	17/09/2004	Sebrae - RJ	Pendente
8	I520 - Java Programming	Computação	40	08/09/2004	21/09/2004	Infnet - Instituto De Formação Internet	Pendente
9	Curso Aberto De Técnicas Para Negociações	Economia	15	30/08/2004	03/09/2004	Sebrae - RJ	Pendente
10	I518 - Projeto De Sistemas E Orientação A O...	Computação	40	23/08/2004	03/09/2004	Infnet - Instituto De Formação Internet	Pendente

Página 1 de 1
 Visualizando registros 1 - 10 de 10

Figura 13 – Tela de Consulta de Cursos – Sistema Horus

CONCLUSÃO

Este trabalho acadêmico visou mostrar as melhores metodologias e tecnologias para o desenvolvimento de um sistema de controle de treinamento, com foco distribuído tanto no seu uso quanto nas plataformas em que pode ser executado.

A gerência de projetos não estimou o custo financeiro direto para o cliente, pois este não existiu. Contudo, foi possível estimar o fator tempo e elencar as tarefas.

O processo de desenvolvimento baseado no RUP especificou os artefatos que devem ser gerados em cada etapa, aumentando a organização e diminuindo o risco de falhas.

A UML proveu os diagramas que embasaram a programação, facilitando e direcionando a etapa de construção do sistema.

A linguagem Java, já conhecida pelo autor da obra e por estar embutida no mesmo paradigma orientado a objetos da análise, facilitou a implementação do software.

O MySQL, por ser gratuito, robusto e com várias funcionalidade proverá as necessidades de persistência do sistema.

Por fim, a arquitetura em três camadas adotada traz portabilidade e facilidade de disseminação ao sistema.

Há a certeza que o sisTreina trará economia para os administradores públicos, praticidade para os usuários e ganho de qualidade no controle feito pelos setores de recursos humanos.

REFERÊNCIAS BIBLIOGRÁFICAS

DATE, C.J. **An Introduction to Database Systems.** . 6ª edição. Addison-Wesley Pub. Co, 1994

DEITEL, Harvey M.; DEITEL, Paul J. **Java: Como Programar.** 4ª edição. Bookman, 2003

GORGES, EDUARDO. **A Lei de Murphy no Gerenciamento de Projetos.** Brasport, 2007

HAGGAR, P. **Java: Guia Prático de Programação.** Editora Campus/O'Really, 2003.

HELLER, P.; Roberts, S. **Guia Completo de Estudos para Certificação em Java.** Editora Moderna, 2004.

IBM. **Rational Unified Process.** Disponibilizado em <http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/WhatIsTheRationalUnifiedProcessJan01.pdf>. Acesso em março/2009

LARMAN, C. **Utilizando a UML e Padrões: Uma Introdução à Análise e Projeto Orientado a Objetos.** Porto Alegre: Bookman, 2000.

MARTINS, JOSÉ C. C. **Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML.** Brasport, 2005.

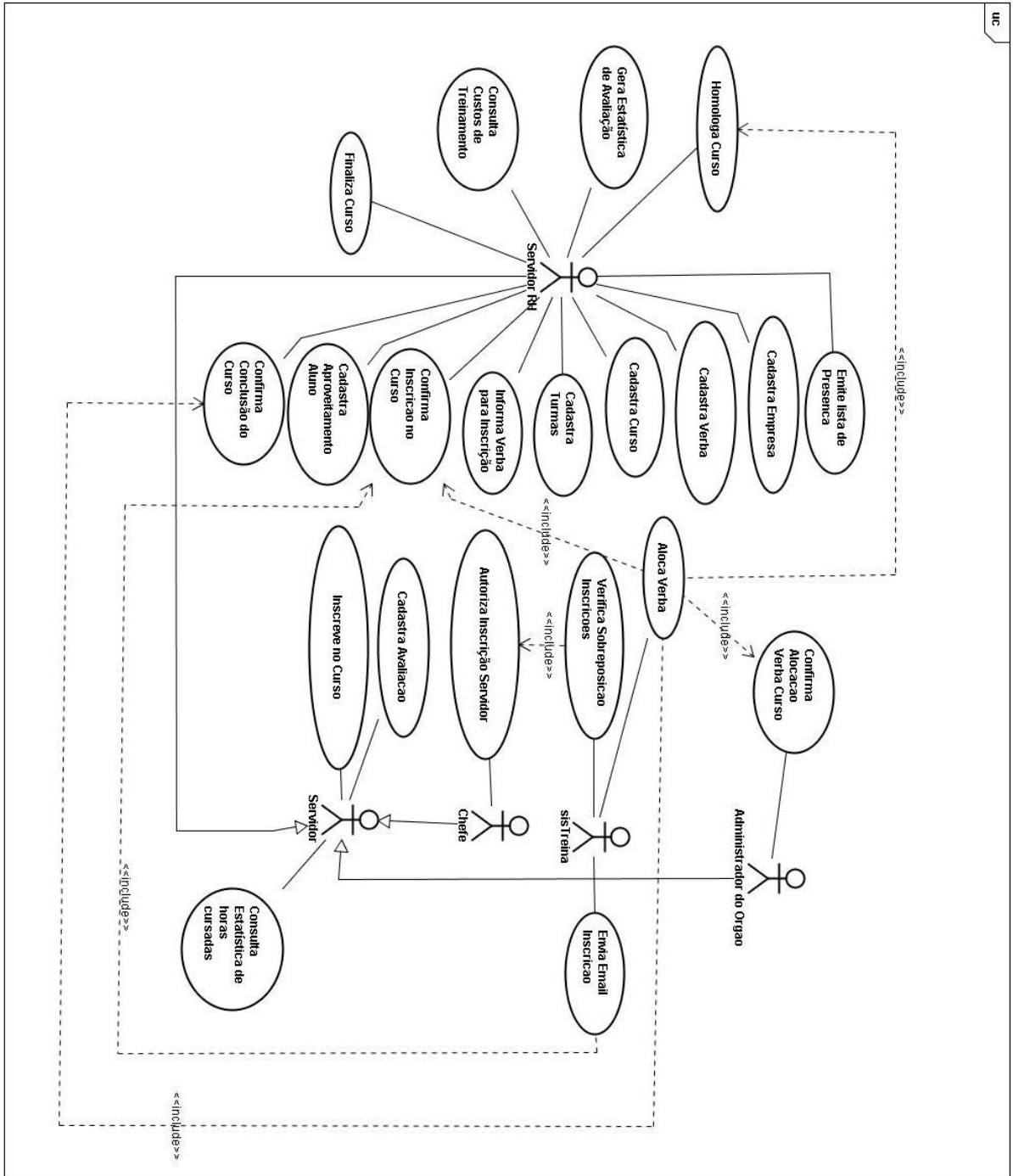
MySQL. **The World's Most Popular Open Source Database.** Disponibilizado em <http://www.mysql.com>. Acesso em janeiro/2009

PRESSMAN, ROGER S. **Engenharia de Software.** 6ª edição. McGraw Hill Brasil, 2006

SEVERINO, Antônio Joaquim. **Metodologia do Trabalho Científico.** 22ª ed. São Paulo: Cortez, 2002

ANEXO – DIAGRAMAS DA UML E GRÁFICO DE GANTT

Figura 14 – Diagrama de Casos de Uso



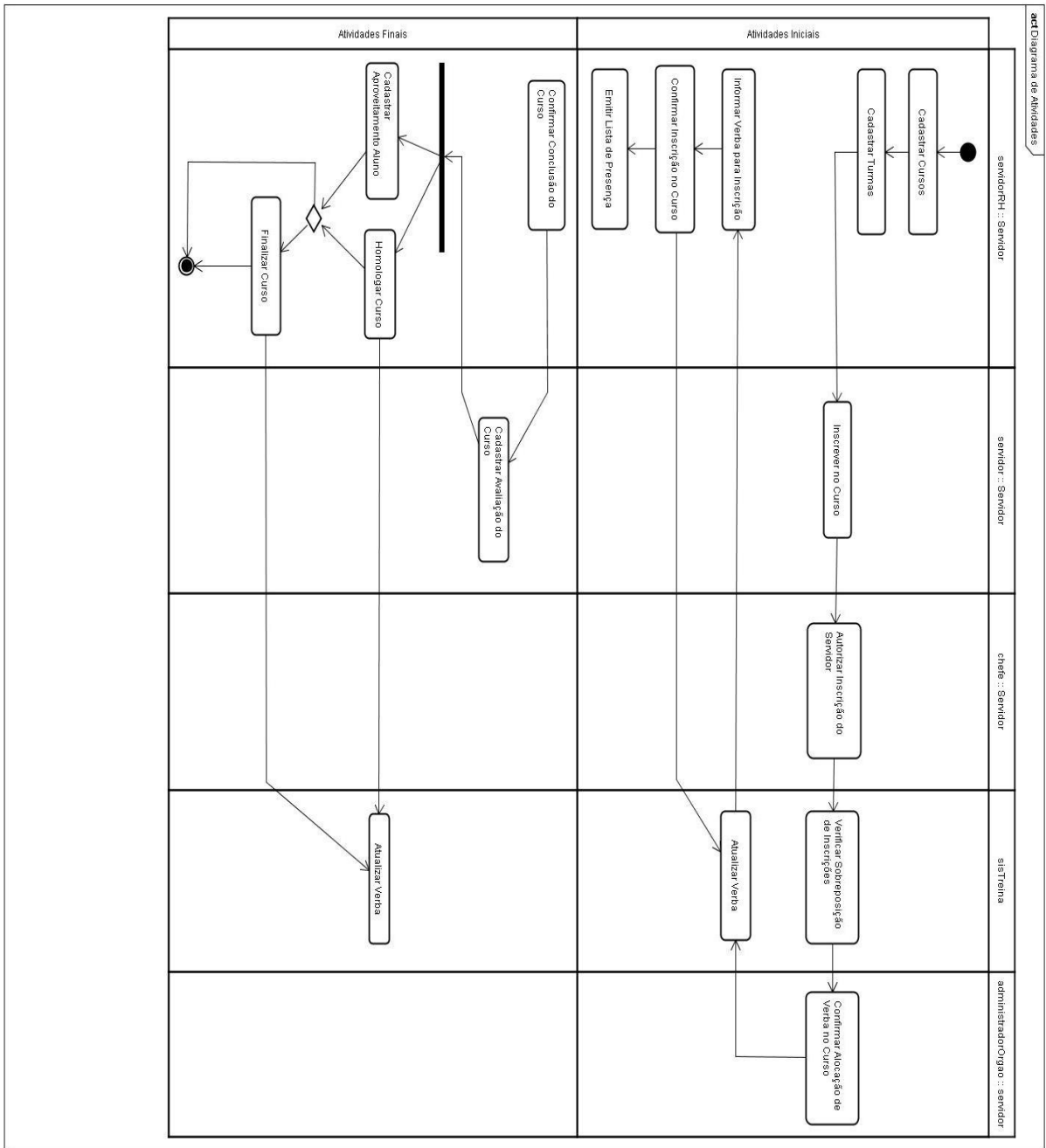


Figura 15 – Diagrama de Atividades

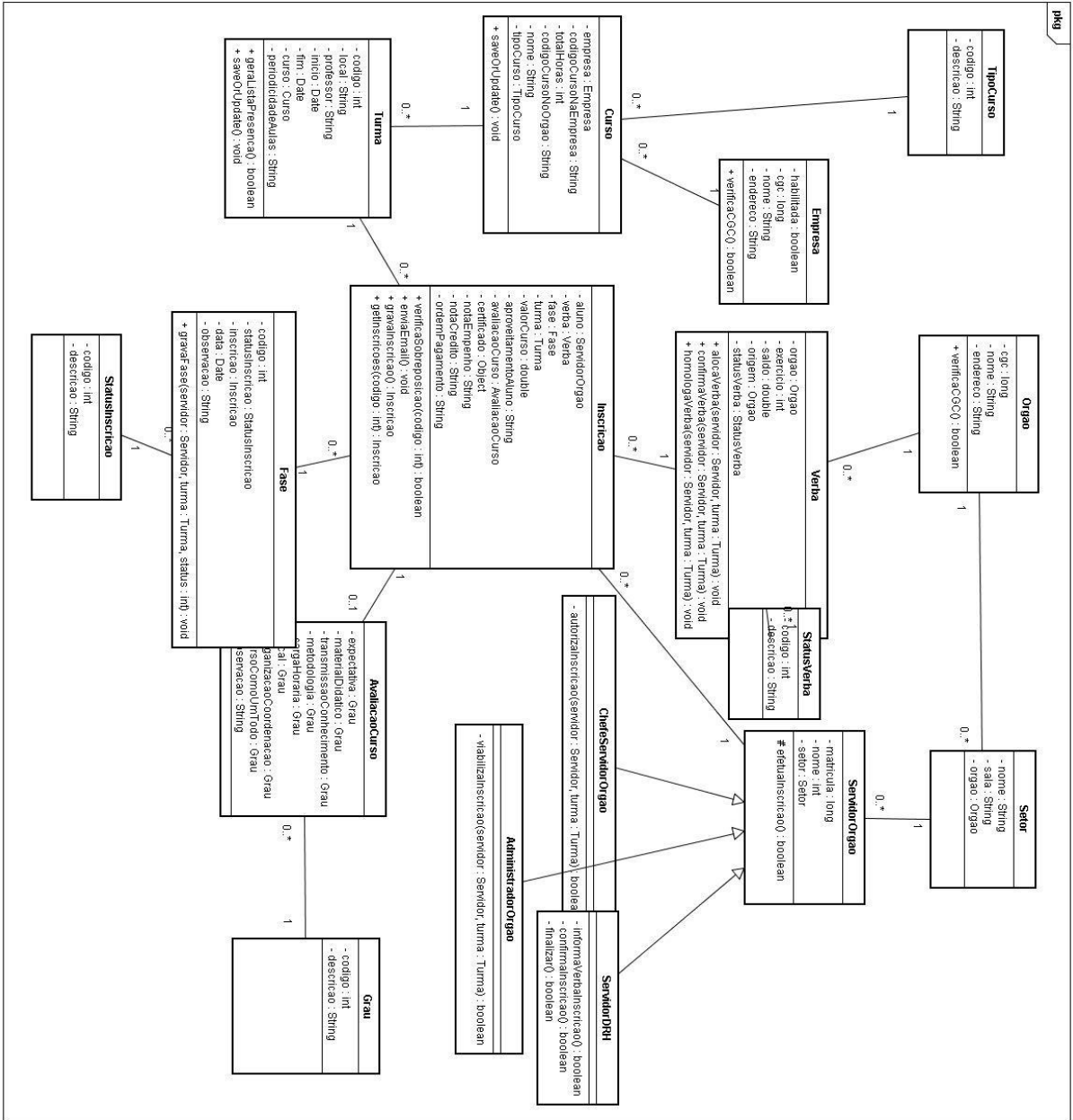


Figura 16 – Diagrama de Classes

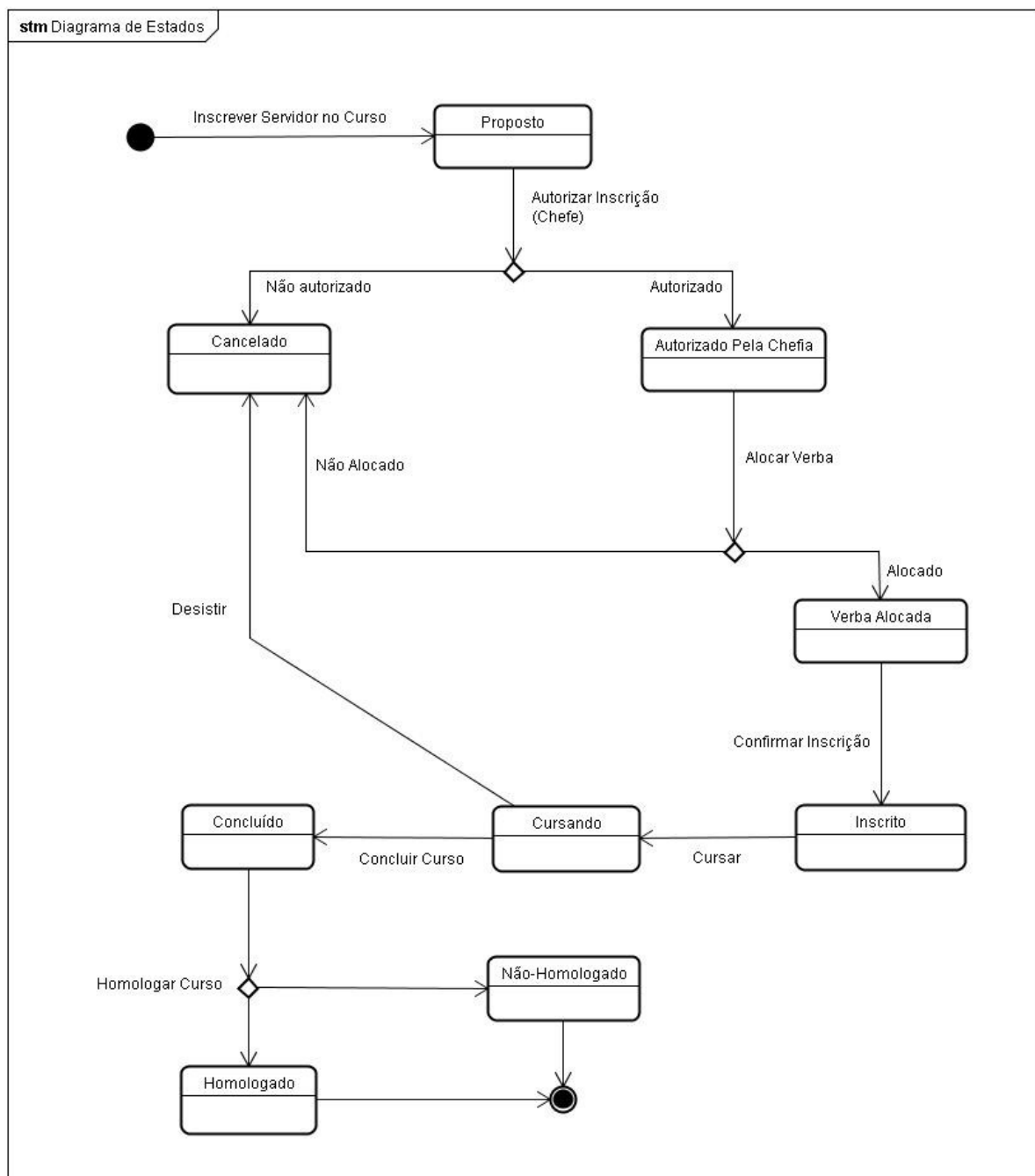


Figura 17 – Diagrama de Estados

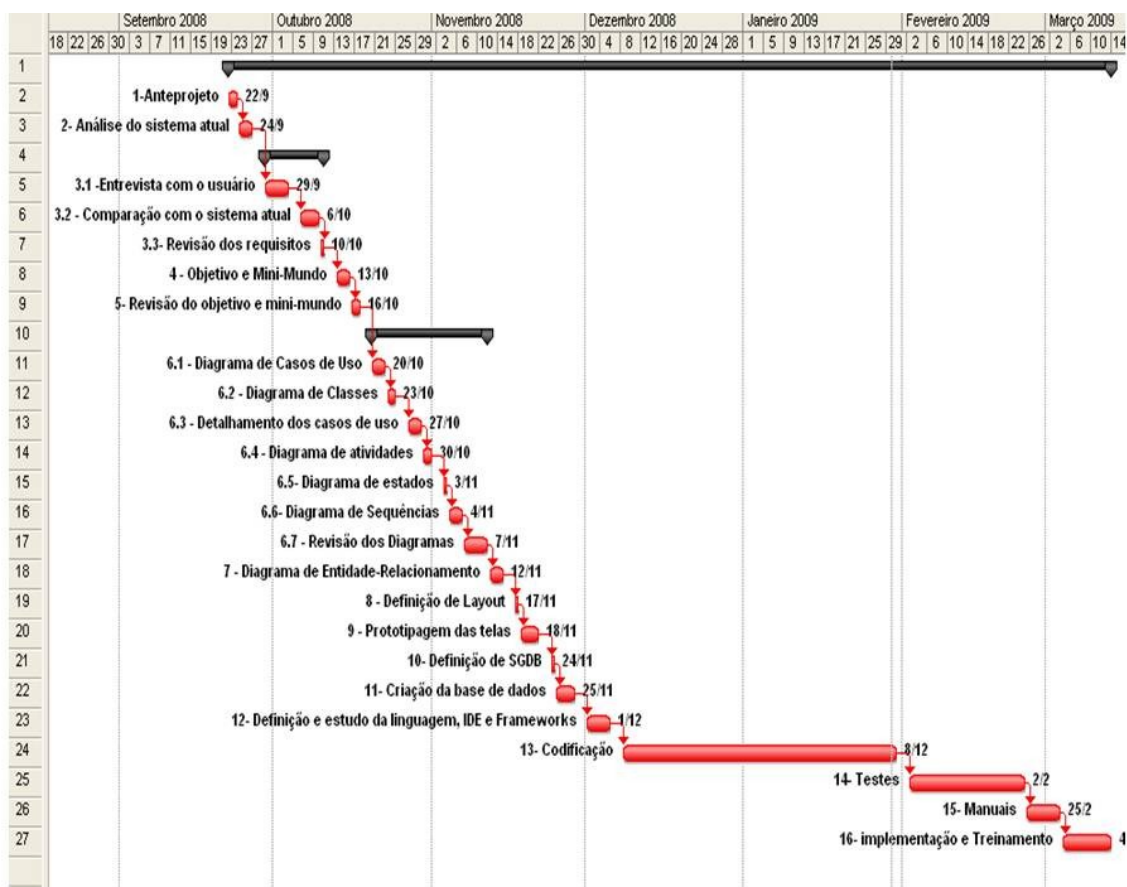


Figura 18 – Gráfico de Gantt