[1] []1 [1]1

LIDIMON CRISTIANO MARTINS ROCHA
NEWTON ASSUNÇÃO DE CAMPOS

METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE* UTILIZANDO RUP E UML

ARAGUARI - MG
2006

LIDIMON CRISTIANO MARTINS ROCHA
NEWTON ASSUNÇÃO DE CAMPOS

METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE* UTILIZANDO RUP E UML

Trabalho de Conclusão de Curso apresentado ao programa de Graduação em Sistemas de Informação da Universidade Presidente Antônio Carlos – UNIPAC, *campus* Araguari como requisito parcial para obtenção do título de Graduado em Sistemas de Informação.

Área de concentração: Engenharia de *Software*.

Orientador: Prof. Rodrigo Videschi

ARAGUARI - MG
2006

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA UNIVERSIDADE PRESIDENTE
ANTÔNIO CARLOS – UNIPAC *campus* ARAGUARI

ROCHA, Lidimon Cristiano Martins; CAMPOS, Newton Assunção
METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE*
UTILIZANDO RUP E UML / Lidimon Cristiano Martins Rocha,
Newton Assunção de Campos. – Araguari, MG;
[s.n.], 2006.

Orientador: Rodrigo Videschi.
Trabalho de Conclusão de Curso - Universidade Presidente Antônio
Carlos, Curso de Sistemas de Informação.

1. *Rational Unified Process*. 2. *Unified Modeling Language*.
3. Desenvolvimento de *Software*. I. Videschi, Rodrigo.
II. Universidade Presidente Antônio Carlos – UNIPAC *campus*
Araguari. Curso de Sistemas de Informação. III. Título

LIDIMON CRISTIANO MARTINS ROCHA
NEWTON ASSUNÇÃO DE CAMPOS

METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE* UTILIZANDO RUP E UML

Trabalho apresentado ao Programa de Graduação em Sistemas de Informação da Universidade Presidente Antônio Carlos – UNIPAC *campus* Araguari, para obtenção do título de Graduado em Sistemas de Informação.

Área de concentração: Engenharia de *Software*.

Banca Examinadora:

Araguari, 12 de Dezembro de 2006.

Prof. Rodrigo Videschi - Unipac

Prof. Leandra Mendes do Vale - Unipac

Prof. Cristiane Lemes Hamawaki - Unipac

Dedico este trabalho a todas as pessoas que eu amo. E obrigado pela compreensão e principalmente pela paciência.

Lidimon Cristiano

Agradecimentos

Agradacemos a todos os nossos companheiros(a) de sala, amigos, familiares e professores que, de alguma forma contribuiram para o fortalecimento deste trabalho.

Os Autores

"Feliz daquele que pode contribuir para a sociedade repassando seus conhecimentos."

Lidimon Cristiano

Resumo

Este trabalho mostra a construção de um projeto de *software* através de um processo de desenvolvimento onde são mostradas todas as fases necessárias de um projeto e os documentos gerados em cada fase conforme o ciclo de vida do produto.

Para isso, foi adotado como metodologia de desenvolvimento o *Rational Unified Process* que prevê quatro fases distintas e dependentes uma da outra. Estas fases são: Concepção, Elaboração, Construção e Transição, onde cada uma possui seu próprio ciclo de atividades, que são realizadas para a passagem de uma fase para outra. Além da metodologia utilizada, será mostrada também toda a modelagem do *software* com o apoio da *Unified Modeling Language*, que é uma linguagem de modelagem unificada e independente de metodologia ou processo. Desta forma, esta metodologia foi aplicada no desenvolvimento de um projeto de *software* real para avaliar a comprovação de sua eficácia.

Palavras-chave: *Rational Unified Process*; *Unified Modeling Language*; Desenvolvimento de *Software*.

Abstract

This document shows the construction of a software project by a methodology of development in which all necessary phases of a project are shown as well as the documents that are generated in each phase according to the life expectancy of the product.

For this, the Rational Unified Process was adopted as a methodology of development which foresees four phases that are distinct from each other but dependent. They are: Conception,Elaboration Construction and Transition and each one has its own cycle of activities that are carried out to the change of the phases. Besides such methodoloy, all the software modeling will also be shown by the support of UML - an unified modeling language that doesn´t depend on process. This way, the methodology was applied to the development of a real software project to evaluate the proof of its efectiveness.

Keywords: Rational Unified Process, Unified Modeling Language, Development of Software.

Lista de Figuras

Lista de Acrônimos

JVM – *Java Virtual Machine*

RUP – *Rational Unified Process*

UML – *Unified Modeling Language*

XP – *eXtreme Programming*

SEI – *Software Engineering Institute*

SCVE – Sistema de Controle de Vendas e Estoque

Sumário

Capítulo 1

Introdução

A crescente demanda das organizações na busca sistemas de informação eficazes e capazes de atender não só as necessidades de usuários finais, mas também para suprir as suas próprias necessidades para a tomada de decisões, fizeram com que as fábricas de *software*[1] passassem a se adequar com metodologias de desenvolvimento e com uma diversificação maior de profissionais, onde cada um fica responsável por uma parte do projeto em questão e ao mesmo tempo respeitando as fases de desenvolvimento baseada na metodologia adotada. Isso é algo que vem sendo praticado desde o surgimento da engenharia de *software*, onde se segue uma metodologia ou modelo de desenvolvimento que tem por objetivo auxiliar os processos de desenvolvimento de *software* para ajudar as empresas a desenvolver ou manter um produto de qualidade, conforme as necessidades do cliente.

Mesmo com várias metodologias que já caíram em desuso e que hoje apenas fazem parte do histórico da engenharia de *software*, estas serviram como *feedback* para o aperfeiçoamento e surgimento de novas metodologias cujo objetivo sempre foi facilitar a construção de *software* com qualidade e de forma sistêmica.

Diante de inúmeras metodologias existentes, o desenvolvimento de *software* sempre foi

[1]Fábricas de *Software* são empresas que possuem um conjunto de recursos, processos e metodologias estruturados de forma semelhante às industrias tradicionais, utilizando as melhores práticas criadas para o processo de desenvolvimento de *software*.

algo complexo onde a sua incorreta aplicação acaba provocando sérios problemas e como conseqüência surge o não cumprimento dos prazos na entrega do produto, ou às vezes o *software* produzido acaba não atendendo as expectativas que o cliente esperava, ou ainda, dificuldade da empresa desenvolvedora em definir um orçamento correto, gerando desta forma uma série de outros problemas provocando até mesmo o fracasso do projeto e causando grande constrangimento e prejuízo tanto por parte do cliente quanto da empresa, esta podendo chegar até mesmo a falência.

Por outro lado, uma metodologia eficaz quando aplicada de forma correta, exige o comprometimento total da equipe desenvolvedora em se adequar à metodologia adotada, que por conseqüência atinge um percentual maior de chance para a conclusão de um desenvolvimento de *software* com êxito, pois com uma metodologia é possível identificar e analisar de forma geral o tamanho e até mesmo a complexidade do sistema e os riscos presentes em cada etapa do desenvolvimento, possibilitando tratá-los logo de início, evitando assim que estes riscos sejam descobertos somente na fase final do projeto. Além da identificação dos riscos, existe uma série de outros fatores e práticas que contribuem para o sucesso de um projeto de *software* que serão vistos no decorrer deste trabalho.

Sendo assim, qualquer metodologia de desenvolvimento de *software* por mais perfeita que seja não irá garantir o sucesso num projeto. Antes de qualquer coisa é preciso ter um *know-how* sobre a metodologia a ser adotada e acima de tudo saber aplicá-la, utilizando todos os artefatos[2] produzidos durante as fases de desenvolvimento como peças fundamentais para chegar a um objetivo proposto.

No curso de sistemas de informação e afins, são vistas várias disciplinas, algumas na verdade são etapas fundamentais de um projeto de desenvolvimento de *software*, porém, tal disciplina é vista de forma muito independente onde não é possível ver um projeto de *software* como um todo. Nas empresas de *software* as etapas de um projeto ou todo o projeto em si, é tido como

[2]Artefato é um documento do resultado de uma ou várias atividades dentro do contexto do desenvolvimento de um *software*. No RUP a cada iteração é produzido um novo documento que servirá de fonte de informação, este documento é chamado de artefato.

um patrimônio da empresa, não sendo divulgado, ou seja, o conhecimento adquirido pertence somente à empresa, não há compartilhamento com a sociedade.

A grande motivação deste trabalho é justamente construir um projeto de *software* através de suas várias etapas usando como metodologia o *Rational Unified Process* (RUP), e que possa servir como referência para aqueles interessados no assunto.

No segundo capítulo deste trabalho serão abordados conceitos sobre metodologias de desenvolvimento de *software*, será citado como exemplo o modelo cascata e o modelo espiral com seus principais problemas quando adotados como metodologia. Será citada também a *eXtreme Programming* (XP), por se tratar de uma metodologia que é utilizada atualmente. O objetivo deste capítulo é fazer com que o leitor possa abstrair os conceitos de metodologias para até mesmo poder comparar com o RUP ou outra metodologia qualquer.

No terceiro capítulo o objetivo será abordar o RUP e suas melhores práticas, que serão utilizadas no decorrer deste trabalho para o desenvolvimento do projeto de *software*.

No quarto capítulo o objetivo principal será abordar o ciclo de vida de um projeto RUP através de suas fases e atividades.

Já no quinto capítulo o objetivo é mostrar a importância da modelagem e uma visão geral da *Unifed Modeling Language* (UML) como ferramenta de apoio fundamental para todas as fases do projeto que será apresentado neste trabalho.

No sexto e último capítulo trata-se essencialmente da aplicação da metodologia RUP junto com o uso da UML para o desenvolvimento do projeto seguindo todas as suas etapas.

Capítulo 2

METODOLOGIAS DE DESENVOLVIMENTO DE *SOFTWARE*

No contexto da engenharia de *software*, uma metodologia é a adoção de um processo[1] de desenvolvimento de *software* qualquer e que, no decorrer de sua utilização, o processo deve possuir uma seqüência de passos ordenados a serem seguidos. Além do processo adotado, uma metodologia é também constituída por técnicas e ferramentas. As técnicas são formas de obterem dados e informações para serem utilizados junto a um processo. As ferramentas facilitam o trabalho e modela o sistema.

Desta forma uma metodologia consiste em dividir todo o contexto de desenvolvimento em várias fases onde cada fase possui um conjunto de atividades com o objetivo de construir ou manter um *software* de qualidade, mas por trás de um *software* de qualidade existe toda uma seqüência de passos que foram seguidos de acordo com o processo adotado pela metodologia. Esses passos são analisados, modelados, construídos, documentados e enfim, validados por alguém para dar continuidade ao processo de desenvolvimento.

[1]No contexto da Engenharia de *Software*, um Processo é uma sequência de passos, onde o objetivo é atingir uma meta.

É importante deixar claro que cada metodologia agrega técnicas, ferramentas e um processo qualquer, que possui sua própria forma de como conduzir um projeto de desenvolvimento de *software* assim como o seu ciclo de vida, ou seja, um sistema computacional pode ser desenvolvido de diversas maneiras, podendo se adotar desde a metodologia com base no modelo[2] cascata até às orientadas a objeto. É claro que, metodologias mais antigas não são mais usadas, mas isso nada impede que novas metodologias sejam criadas com base em suas antecessoras.

Uma metodologia quando adota o processo RUP, acaba se tornando uma metodologia flexível, isso significa que se pode customizar e adequá-la de acordo com a necessidade ou tamanho do projeto. No caso do RUP que é um processo iterativo [2], ele suporta grandes e complexos projetos com uma gama de profissionais envolvidos, podendo ser configurável para pequenos projetos resultando em uma redução de artefatos não aplicáveis a estes pequenos projetos. O RUP será visto com mais detalhes no próximo capítulo sendo a base principal para o desenvolvimento deste trabalho.

Existem diferenças entre uma metodologia e outra, o Modelo Seqüencial Linear conhecido mais como Modelo Cascata ou Ciclo de Vida Clássico, foi o modelo mais amplamente usado pela engenharia de *software* e o mais antigo[3]. É uma metodologia diferente do RUP, que propõe um desenvolvimento de forma linear. Diversas metodologias seguem um padrão comum de desenvolvimento (análise, projeto, codificação, teste), mas com um ciclo de atividades diferente.

Existe também o modelo de desenvolvimento rápido de aplicação (*Rapid Application Development*, RAD) que é uma metodologia de desenvolvimento incremental que enfatiza um ciclo de desenvolvimento extremamente curto, porém utilizam-se os princípios do ciclo de vida clássico.

Outra metodologia que pode ser citada é o *eXtreme Programming* (XP) que vem sendo adotada na Europa, Estados Unidos e recentemente no Brasil[4]. O XP é uma metodologia para equipes pequenas, de preferência até 12 membros, é considerada leve em relação ao RUP por não produzir a mesma formalidade para a geração de artefatos. Esta metodologia propõe

[2]Modelo tem o mesmo significado de Processo, ou seja, é uma seqüência de passos a ser seguido afim de obter um resultado.

um desenvolvimento de forma rápida e ágil. Inclusive a programação é feita em par, realizando um revezamento entre os programadores e logo no início do projeto o sistema já começa a ser implementado. Esta metodologia será abordada com um pouco mais de detalhes durante os próximos tópicos deste capítulo.

Contudo, percebe-se que as metodologias de desenvolvimento de *software* possuem objetivos comuns, e com o tempo, novas técnicas são aprofundadas para melhorar ou criar novos processos.

2.1 Modelo Artesanal de Desenvolvimento de *Software*

Os primeiros *softwares* eram desenvolvidos de forma bem artesanal, onde o responsável pela construção do programa adquiria os conhecimentos necessários através de outro desenvolvedor, nesta época não havia uma bagagem sobre desenvolvimento de *software*. Prevalecia o talento pessoal, a criatividade e inspiração, o que significa um desenvolvimento de *software* fortemente de cunha pessoal. O produto era uma obra de arte, só quem desenvolveu tinha conhecimento e podia alterá-lo.

A Figura 2.1 demonstra como ocorria o desenvolvimento: construía a primeira versão, submetia ao cliente, realizava os consertos e refinamentos, buscava novamente a aprovação do cliente, caracterizando um trabalho do tipo tentativa e erro.

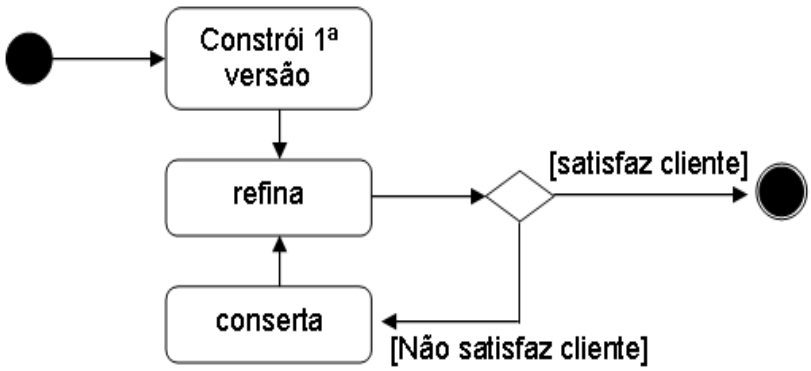

Figura 2.1: Modelo Artesanal.

Essa forma de desenvolver era também muito limitada, pois a qualidade dependia de quem estava desenvolvendo, a produção em escala era difícil, a manutenção era totalmente dependente do criador, não era aplicada nenhuma prática da engenharia de *software* e a qualidade dependia diretamente do talento individual, ou seja, cada desenvolvedor tinha sua própria metodologia de como desenvolver.

Com o passar do tempo houve a necessidade de se criar maneiras de desenvolver *software*, onde o desenvolvedor poderia se apoiar a um modelo de desenvolvimento para produzir um sistema com mais qualidade. Os tópicos 2.2 e 2.3 mostram dois modelos de desenvolvimento como exemplo, que foram adotados a fim de se evitar o trabalho artesanal e padronizar de certa forma o desenvolvimento de sistemas.

2.2 Modelo Cascata

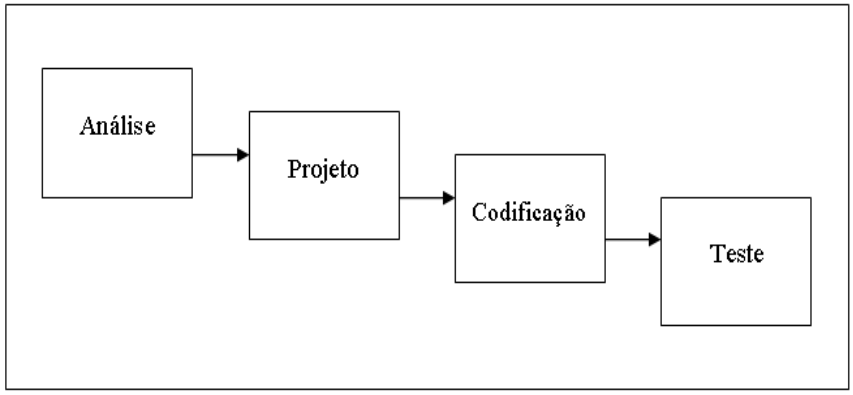

Figura 2.2: Modelo Cascata.

O Modelo Cascata ou Ciclo de Vida Clássico foi proposto por Royce[5] em 1970 sendo o único modelo com uma aceitação geral no mercado de desenvolvimento até a década de 1980. Este modelo foi adotado como uma metodologia padrão de desenvolvimento de *software*, servindo como uma referência para muitos outros modelos e para projetos modernos.

O modelo cascata tem como foco principal o desenvolvimento de *software* de maneira seqüen-

cial onde passagem de uma fase para outra só é realizada quando a fase anterior estiver totalmente pronta, e assim sucessivamente. Este modelo não permitia voltar em uma fase anterior para fazer alterações, a abordagem partia do princípio de que o cliente participava ativamente da fase de requisitos e que sabia muito bem o que queria não sendo necessário a alteração dos requisitos. Este modelo foi um avanço muito grande para a área de desenvolvimento de sistemas, mas tornou-se limitado devido a alguns problemas que comprometia o desenvolvimento.

2.2.1 Problemas do Modelo Cascata

De acordo com Pressman[3], o Modelo Cascata quando aplicado ao desenvolvimento de *software*, apresenta os seguintes problemas:

- Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Apesar de o 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 seqüencial linear exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de vários projetos.
- O cliente precisa ter paciência. Uma versão executável do programa não vai ficar disponível até o projeto terminar. Um erro grosseiro pode ser desastroso, se não for detectado até que o programa executável seja revisto.

Conforme os problemas mencionados por Pressman[3], este é um modelo que não permite iterações, ou seja, se algum requisito na fase inicial foi entendido de forma errada pelo analista, este erro só será descoberto na fase final de implementação, pois o modelo é uma cascata que não volta para trás. Desta forma não fornece *feedback*, não suporta modificações nos requisitos entre as fases, o que possibilitaria revisões e principalmente a descobertas de riscos. Assim um dos grandes problemas desta abordagem é que ela oculta os riscos e quando descobertos torna caro desfazer erros de fases anteriores. É como comparar a construção de um prédio, foi projetado

um prédio de cinco andares, mas após a construção da estrutura principal, descobriu-se que a estrutura suporta apenas três andares, sendo assim, desfazer toda a estrutura já construída torna-se um custo muito alto. Esse é um dos principais motivos que levam as fábricas de *software* a buscar por metodologias eficientes para que se defina um caminho a ser percorrido, garantindo desta forma a qualidade do projeto tanto em termos de produto final quanto orçamento fixo e cronograma.

Com o objetivo de contornar e oferecer uma melhor forma de desenvolvimento, surgiram outros modelos, um deles é o Modelo Espiral que será visto a seguir.

2.3 Modelo Espiral

O Modelo Espiral foi proposto por Boehm[5] em 1988. É uma evolução do modelo em cascata combinado com processos de natureza iterativa e incremental. O desenvolvimento baseado no Modelo Espiral é composto de uma série de versões incrementais. Primeiramente é desenvolvido um ciclo que corresponde às atividades definidas no modelo conforme a Figura 2.3, desta forma é gerada a primeira versão do sistema. Novas iterações eram realizadas produzindo um produto cada vez mais complexo. Neste modelo foram também definidas as atividades, onde o objetivo era garantir a qualidade do produto final, as atividades deste modelo eram as seguintes:

- **Comunicação com o cliente**: nesta tarefa o objetivo era estabelecer todas as necessidades do cliente.
- **Planejamento**: definição dos recursos, prazos, custos e outras informações relacionadas ao projeto.
- **Análise de Riscos**: avaliação dos riscos técnicos e gerenciais do projeto.
- **Engenharia**: procedimentos necessários para construção de uma ou mais representação do sistema.

- **Construção e Entrega**: etapas fundamentais para construir, testar, instalar e fornecer suporte ao usuário.

- **Avaliação pelo cliente**: *feedback* do cliente baseado na avaliação das versões criadas e fornecidas durante o estágio de engenharia e implementação.

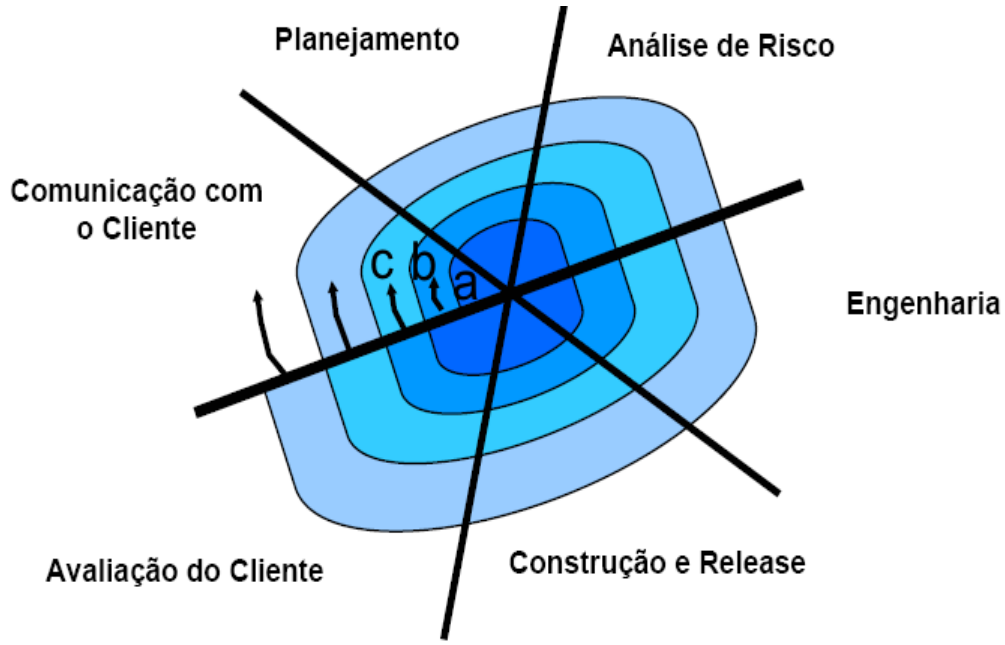

Figura 2.3: Modelo Espiral.

Se o modelo for comparado com o modelo cascata, percebe-se que o modelo cascata só disponibilizava uma versão executável do sistema no fim do projeto e no modelo espiral é disponibilizada uma versão executável a cada iteração, o que garante mais qualidade no *software*.

No Modelo Espiral de desenvolvimento, os desenvolvedores movem-se em volta da espiral no sentido horário. A primeira versão do produto inicia no ponto 'A' conforme mostrado na Figura 2.3, neste ponto são executadas as atividades de comunicação com o cliente, resultando no desenvolvimento das especificações do produto. Na atividade de planejamento são definidos os recursos e outros ítens necessários na execução do projeto, a atividade seguinte é a análise de riscos e a representação de como construir ocorre na atividade de engenharia, em seguida vem

à atividade de construção de liberação do produto para o cliente, finalmente o cliente realiza a avaliação do produto fornecendo informações sobre o desempenho do mesmo. No ponto 'B' da Figura 2.3 o processo se repete gerando uma nova versão do produto. O *software* evolui à medida que o processo avança, o desenvolvedor e o cliente entendem melhor e reagem aos riscos em cada nível.

Mesmo com o avanço em relação ao modelo cascata, o modelo espiral também apresentava alguns problemas.

2.3.1 Problemas do Modelo Espiral

As dificuldades inerentes à aplicação do Modelo Espiral também são causadas pela sua natureza genérica e pelo forte apoio na análise de riscos para obter sucesso, pois:

- É preciso contar com uma equipe experiente na identificação e redução de riscos, pois se no primeiro ciclo os requisitos não foram entendidos conforme a realidade, os demais ciclos serão implementados e aperfeiçoados com base na primeira iteração, ou seja, todos os ciclos posteriores incorporarão o erro inicial.
- O custo do projeto é elevado por contar com uma equipe experiente.
- O modelo necessita de uma maior elaboração de seus passos para que possa ser aplicado de maneira consistente por toda a equipe de desenvolvimento. Principalmente se a equipe for grande e contiver membros inexperientes.
- A negociação de contrato com um cliente externo é difícil, pois a espiral requer certo nível de flexibilidade e liberdade nos prazos e custos do desenvolvimento.

Pode-se notar que, partindo do modelo artesanal depois passando pelo modelo cascata até chegar ao modelo espiral, é possível perceber as melhorias que cada modelo teve em relação ao seu antecessor. No próximo tópico será visto uma metodologia utilizada atualmente para o desenvolvimento de *software*.

2.4 *eXtreme Programming* (XP)

Segundo Teles [4], *eXtreme Programming*, ou XP, é um processo de desenvolvimento que busca assegurar que o cliente receba o máximo de valor de cada dia de trabalho da equipe de desenvolvimento. Ele é organizado em torno de um conjunto de valores e práticas que atuam de forma harmônica e coesa para assegurar que o cliente sempre receba um alto retorno do investimento em *software*. A XP além de valorizar bastante o cliente, é um processo voltado para:

- Projetos cujos requisitos são vagos e mudam com freqüência.
- Desenvolvimento de sistemas orientados a objeto.
- Equipes pequenas, preferencialmente até 12 desenvolvedores.
- Desenvolvimento incremental (ou iterativo), onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do tempo.

A XP foi criada por Kent Beck no final dos anos 90 [4], é um processo que se enquadra no Manifesto Ágil[3] de desenvolvimento com foco na agilidade da equipe e qualidade do projeto, valorizando os seguintes princípios:

- Indivíduos e interações ao invés de processos e ferramentas.
- *Software* funcional ao invés de documentação extensa.
- Colaboração com clientes ao invés de negociação de contratos.
- Responder a mudanças ao invés de seguir um plano.

Todas as metodologias que valorizam os principios citados anteriormente, são considerados processos ágeis. Além do XP, existem também outras metodologias como Crystal[4] que adotam

[3]Manifesto Ágil de Desenvolvimento ou Processos Ágeis . Maiores informações podem ser obtidas em http://www.agilemanifesto.org.

[4]Maiores informações podem ser obtidas em http://www.crystalmethodologies.org.

os mesmos princípios. A XP é também composta por um conjunto de valores e práticas que devem ser adotados por qualquer empresa que queira utilizar a XP para o desenvolvimento de *software*. Os valores se resumem em:

- ***Feedback***: é necessário que o cliente sempre esteja participando do desenvolvimento para reavaliar parte do produto desenvolvido, desta forma o cliente aprende com o pouco que foi desenvolvido e vai realimentando a equipe de acordo com suas necessidades.

- **Comunicação**: tem como prioridade o uso do diálogo para que todos do projeto possam compreendê-lo, a idéia é que a comunicação seja feita face a face com o mínimo de documentação.

- **Simplicidade**: tem por objetivo garantir que a equipe faça apenas o que realmente é necessário, ou seja, implementar as necessidades do cliente de forma mais simples possível para que o mesmo possa utilizar e prover *feedback* afim de evitar especulações por parte da equipe.

- **Coragem**: durante o desenvolvimento, a equipe faz de forma contínua a manutenção do *software* e ao mesmo tempo cria novas funcionalidades, sendo que em muitos casos é necessário alterar o que já estava funcionando e isso conseqüentemente gera riscos e falhas no sistema.

Uma equipe que utiliza a XP precisa ter coragem e acima de tudo acreditar nas práticas e valores da XP para superar estas etapas que são consideradas comuns a todo o momento.

Além dos valores, a XP se baseia em práticas conforme ilustrado na Figura 2.4, assim como o RUP que também se baseia em outras práticas que serão vistas no próximo capítulo. Porém a XP se diferencia completamente em relação ao RUP. Na XP, a geração de documentos deve ser mínima, tudo é feito pensando no presente sem se preocupar com o futuro. O cliente, por exemplo, através do seu *feedback* garante o andamento do projeto ao invés estar seguindo um documento com os requisitos necessários.

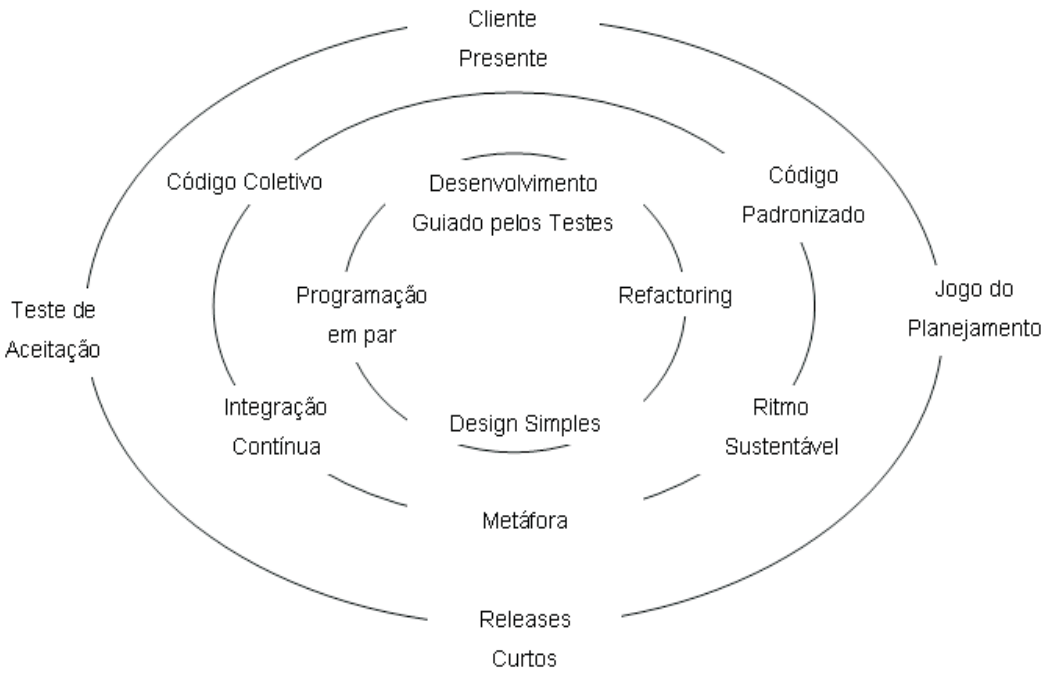

Figura 2.4: Práticas adotadas pela XP.

Portanto o objetivo deste tópico foi apenas citar a XP de forma bem resumida sendo que não foram citadas vantagens e nem desvantagens da XP, pois se trata de uma metodologia em crescimento, sendo adotada por algumas empresas do ramo de desenvolvimento de *software*[6].

Capítulo 3

RATIONAL UNIFIED PROCESS (RUP)

O RUP é um processo de engenharia de *software*. Ele fornece uma abordagem disciplinada para assumir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de *software* de alta qualidade que satisfaça as necessidades de seus usuários finais dentro de prazo e orçamento previsíveis. [2].

Adotar o processo RUP como uma metodologia padrão de desenvolvimento, significa trazer qualidade e organização tanto para a empresa que desenvolve quanto para o produto que está sendo desenvolvido, pois o RUP define como as tarefas devem ser feitas, define o papel de cada membro da equipe e ainda atribui a cada um deles responsabilidades dentro do projeto.

O RUP em geral atende a grandes projetos de *software*, mas nada o impede de se adequá-lo a pequenos projetos. O RUP gera artefatos durante todo o ciclo de desenvolvimento, o que ajuda a manter *feedback*, controle sobre o processo e prever riscos, além disso, o RUP agrega em seu processo a Linguagem de Modelagem Unificada (UML) para modelar, gerar artefatos, especificar e ainda utiliza as melhores práticas de desenvolvimento de *software* que serão vistas no tópico 3.1 deste capítulo.

Atualmente o RUP é desenvolvido e mantido pela Rational[1], uma empresa IBM. O RUP é um processo atualizável e adaptável de acordo com as necessidades da empresa que o utiliza.

A Rational possui diversas ferramentas para serem utilizadas junto com o RUP a fim de ganhar mais produtividade principalmente quando se trata de uma equipe muito grande, pois o processo necessitará de uma formalidade maior.

Para utilizar o RUP como metodologia de desenvolvimento, não é necessário a aquisição de nenhuma ferramenta da Rational, além disso, estas ferramentas possuem um custo muito alto, apesar de serem bastante úteis em projetos complexos que envolvem varias equipes até mesmo espalhadas geograficamente.

É importante deixar bem claro que, dentro de uma metodologia existe um processo adotado e que será seguido, neste trabalho foi adotado o RUP, no qual foi feito um estudo sobre grande parte do processo e ao mesmo tempo o RUP foi adequado conforme o tamanho e as necessidades deste projeto, que será apresentado no Capítulo 6.

A documentação do RUP vem integrada junto com a ferramenta IBM *Rational Method Composer*, veja a Figura 3.1. Esta ferramenta é utilizada para customizar o RUP dentro de uma empresa de desenvolvimento, ou seja, todos os projetos seguiram um padrão básico. Pode haver também a necessidade customizar uma versão do RUP somente para um projeto epecifico, tudo vai depender da complexidade do sistema a ser desenvolvido. O RUP pode ser visualizado através de qualquer navegador web como mostra a Figura 3.2.

O RUP adota em seu processo as melhores práticas de desenvolvimento de *software* que serão vistas a seguir.

3.1 Melhores Práticas de Desenvolvimento adotadas pelo RUP

As melhores práticas de desenvolvimento de *software* se referem a uma abordagem dos princípios já utilizados no desenvolvimento de *software* que já levaram projetos ao sucesso, ou seja, são práticas usadas na indústria por organizações de sucesso. A seguir será mostrada um estudo sobre as melhores práticas utilizadas pelo RUP e consagradas pelo mercado de projeto de

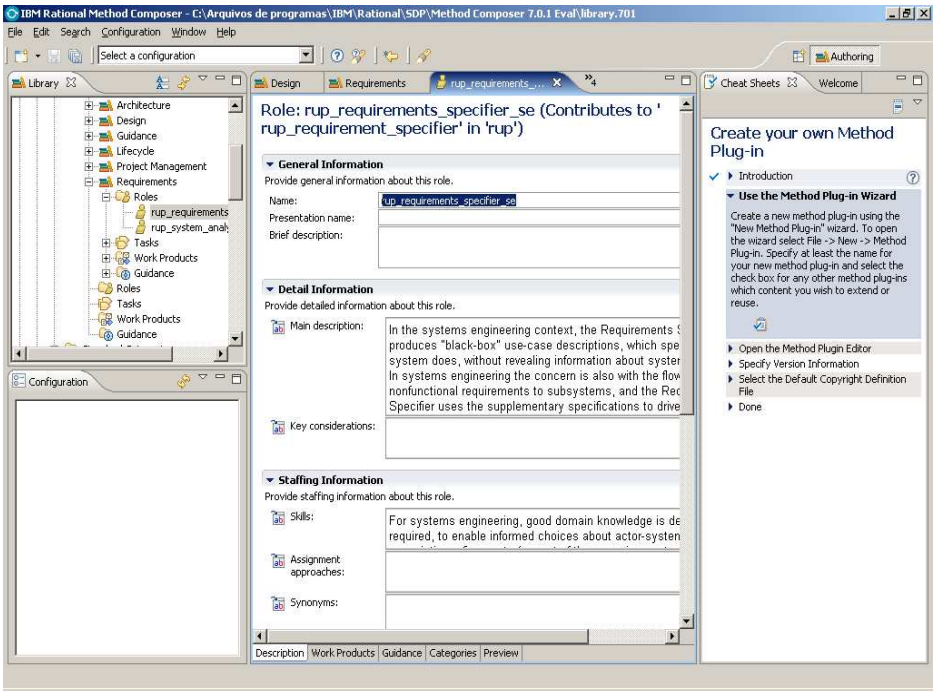

Figura 3.1: *Rational Method Composer*, ferramenta para customizar o RUP.

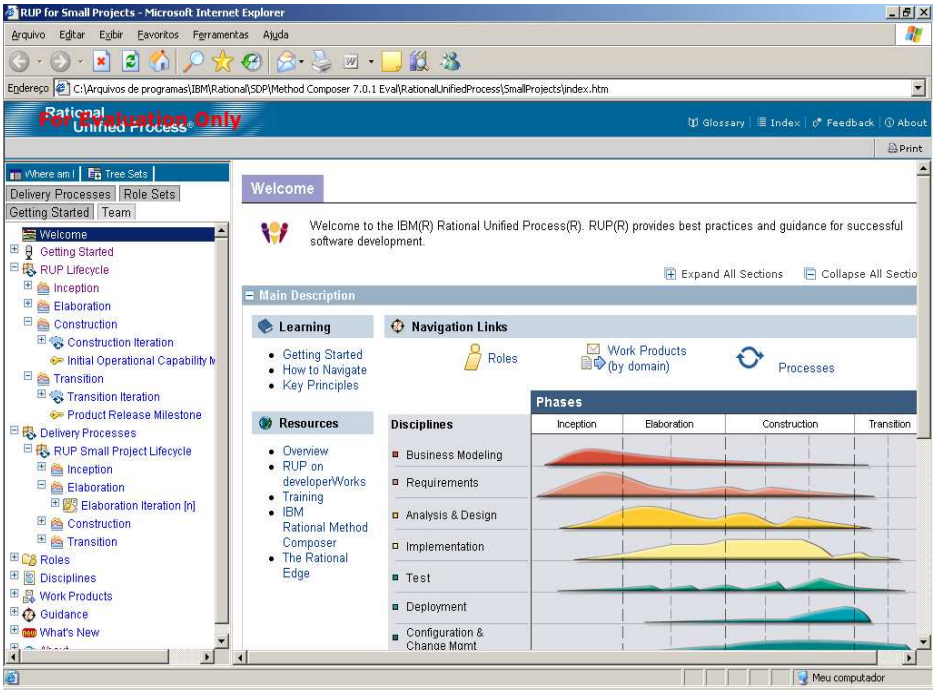

Figura 3.2: Documentação do RUP via *browser*.

software.

3.1.1 Desenvolvimento Iterativo

As metodologias clássicas de desenvolvimento de aplicativos tendem a mascarar as falhas de projeto, que são descobertas somente na fase final de desenvolvimento. Isto provoca um custo muito alto ou a inviabilidade de correção do projeto, tendo como conseqüência o cancelamento do mesmo.

O processo iterativo e incremental inicia o projeto com a convivência das falhas e suas soluções tempestivamente. Segundo Kruchten[2] uma abordagem iterativa consiste em construir um pouco, implementar um pouco, projetar um pouco, validar e então assumir mais requisitos, construir um pouco mais, implementar um pouco mais e assim por diante, até que tenha acabado. No RUP, as vantagens do desenvolvimento iterativo são:

- Os riscos são reduzidos mais cedo.
- Os requisitos variáveis são melhor gerenciáveis.
- A capacidade de reutilização é aumentada.
- Os diversos membros da equipe de desenvolvimento trabalham de forma uniforme.
- Os testes são utilizados desde o início do projeto.
- As mudanças são melhores acomodadas.

3.1.2 Gerenciamento de Requisitos

Requisito é definido como uma condição ou uma capacidade com o qual o sistema deve estar em conformidade. Os requisitos de um sistema em desenvolvimento assumem o aspecto dinâmico, ou seja, pode sofrer mudança durante o projeto. O gerenciamento de requisito consiste em localizar, organizar e controlar as mudanças dos requisitos variáveis.

As empresas devem desenvolver as seguintes habilidades para gerenciar de forma eficiente os requisitos:

- Análise do problema em alto nível, definindo as fronteiras da solução e as restrições de negócios dessa solução.
- Noções básicas sobre as necessidades dos envolvidos. Realizar o levantamento das informações necessárias para ter uma visão do problema a partir dos envolvidos com o negócio da empresa. As necessidades devem ser formalizadas através de artefatos.
- Definição do sistema por meio da conversão e organização do entendimento das necessidades do cliente.
- Gerenciamento do escopo do projeto que consiste em definir a abrangência do sistema baseado nas necessidades do cliente, desenvolvendo o sistema gradativamente e escolhendo cuidadosamente os requisitos a serem incrementados.

No RUP, o gerenciamento de requisitos é controlado com base nos artefatos gerados, ou seja, sempre quando houver mudança de requisitos tanto em uma iteração quanto em uma fase, o artefato é revisado ou atualizado para uma nova versão. Por outro lado o gerenciamento de requisitos pode não ocorrer em projetos pequenos que possuem poucos requisitos.

3.1.3 Arquitetura baseada em Componentes

Um componente em *software* pode ser um módulo, um pacote ou um subsistema com uma função clara, fronteira definida e pode ser integrado numa arquitetura maior. A criação de sistema baseado numa arquitetura de componentes possibilita reduzir o tamanho efetivo e a complexidade do projeto, tendo como conseqüência um sistema mais robusto e flexível.

A construção de sistema baseado em uma arquitetura de componentes possibilita a reutilização de componentes de terceiros sendo que, estes componentes já foram testados e aprovados em outros projetos. O projeto torna-se modularizado e os componentes são encapsulados ao sistema.

Na verdade quando se modela um sistema, automaticamente defini-se a sua arquitetura, pois não adianta nada pensar em uma arquitetura sem antes fazer um modelo. Uma arquitetura pode ser representada por um diagrama que mostra a conexão entre componentes e seus relacionamentos com outros.

3.1.4 Modelagem Visual

Devido à complexidade da realidade, o ser humano aprendeu a desenvolver modelos para ter uma visão simplificada e entender o todo. Nos projetos de *software* não é diferente, simplifica-se o sistema através da modelagem utilizando alguma linguagem padrão como a UML (*Unified Modeling Language*). Com a utilização da UML tem-se uma melhor comunicação entre a equipe de desenvolvimento, captura dos requisitos com precisão, ajuda na compreensão de sistemas complexos e gerenciamento dos modelos. A UML será abordada com mais detalhes no quinto capítulo.

3.1.5 Verificação Contínua da Qualidade do *Software*

A Figura 3.3 apresenta a relação custo de correção de um projeto à medida que se desenvolvem as várias fases do mesmo. O custo vai aumentando com o tempo até chegar o momento que a correção de alguma falha se torna inviável.

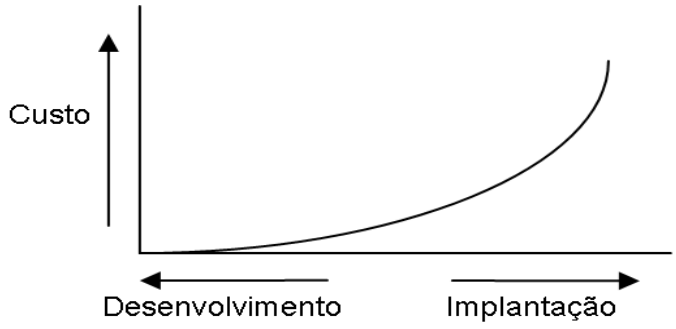

Figura 3.3: Custo para resolver problemas conforme o desenvolvimento vai progredindo.

O gerenciamento da qualidade tem como finalidade identificar métricas adequadas para

servir como padrão a ser utilizada no projeto para descobrir desvios o mais cedo possível. O gerenciamento da qualidade é implementado durante todo o ciclo de vida do projeto.

3.1.6 Controle de Mudanças do *Software*

O controle de mudanças envolve o estabelecimento de procedimentos que podem ser repetidos, tornando a coordenação das atividades, dos artefatos e da equipe envolvida administráveis. O controle sobre as mudanças oferece soluções para vários problemas do projeto de *software*:

- A propagação de mudança é avaliável e controlada.
- Mudanças de requisitos são definidas e repetíveis.
- As mudanças podem ser mantidas num sistema robusto e personalizável.

3.2 Conceitos-chave do RUP

O RUP não é composto apenas pelas melhores práticas de desenvolvimento. Ele possui conceitos básicos como: disciplina, atividade, papel, artefato e fluxo de trabalho. A Figura 3.4 representa de forma geral estes conceitos que são baseados em quem está fazendo, o quê, como e quando, conforme descrito em um processo.

3.2.1 Disciplina

A disciplina é um conjunto de atividades inter-relacionadas a um assunto específico, ou seja, é uma seqüência semi-ordenada, onde são realizadas atividades para alcançar um determinado resultado. Um exemplo de seqüência destas atividades poderia ser primeiramente a elaboração do documento visão, depois documento glossário e assim por diante. Cada disciplina além de possuir as atividades, possui também os papéis responsáveis por cada artefato.

No RUP quando uma seqüência de disciplinas é percorrida em uma única fase, tem-se uma iteração. Uma fase pode ter mais de uma iteração, ou seja, pode-se repetir, tudo vai depender

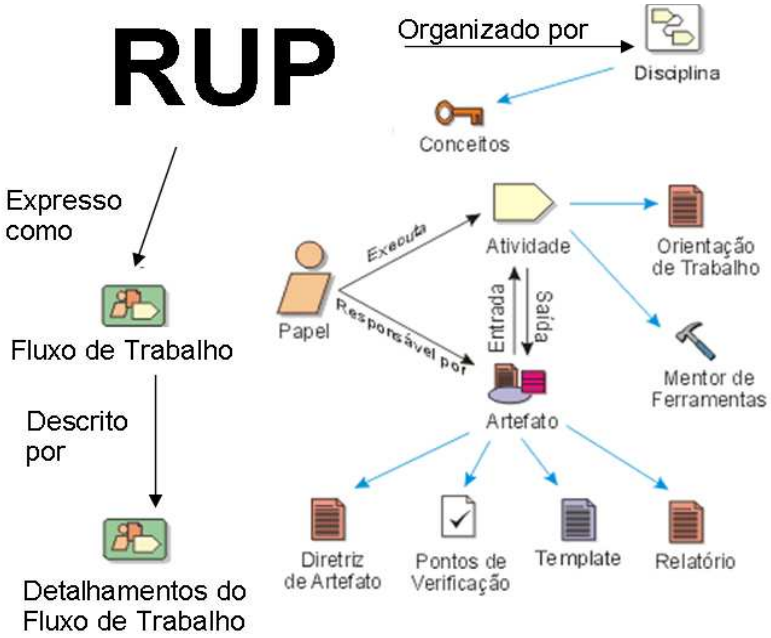

Figura 3.4: Conceitos-chave do RUP[1]

da necessidade do projeto.

O objetivo de agrupar as atividades em disciplina é facilitar a compreensão do projeto.

3.2.2 Fluxo de Trabalho

O 'quando': Um processo é constituído pela iteração entre os papéis, quais artefatos são utilizados como entrada e saída e a seqüência das atividades que produzem um resultado significativo para a fase ou o projeto. Esta seqüência de atividades recebe o nome de fluxo de trabalho. Os fluxos são centrados no processo RUP e divididos em seis fluxos centrados na engenharia e três centrados no suporte. Os fluxos de engenharia são:

- Fluxo de Modelagem de Negócio.
- Fluxo de Requisitos.
- Fluxo de Análise e *Design*.
- Fluxo de Implementação.

- Fluxo de Teste.
- Fluxo de Implantação.

Os fluxos de suporte são:

- Fluxo de Gerenciamento de Projeto.
- Fluxo de Configuração e Gerenciamento de Mudança.
- Fluxo de Ambiente.

O detalhamento dos diversos fluxos estão descritos no capítulo 4.

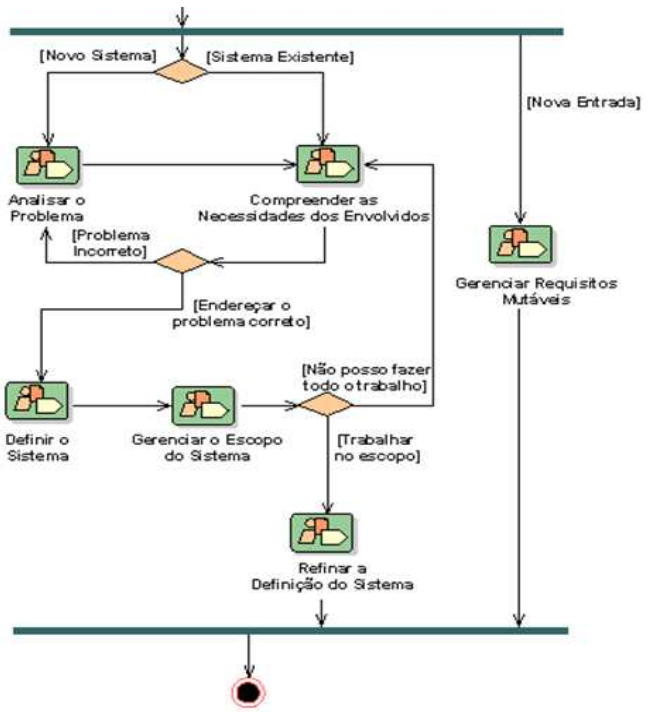

Figura 3.5: Exemplo de fluxo referente à disciplina de requisitos do RUP.

No RUP representa-se o fluxo de trabalho pelo diagrama de atividade conforme exemplo da Figura 3.5, que expressa o detalhamento do fluxo de trabalho mostrando os papéis envolvidos, os artefatos de entrada e saída e as atividades executadas.

3.2.3 Papel

O 'Quem': No RUP o indivíduo tem determinado comportamento e responsabilidade, denominado papel, assim como um ator de novela. Um papel é um conjunto de atividades coerentemente executadas, onde os artefatos são gerados ou modificados. O papel é desempenhado por uma pessoa ou grupo de pessoas formando uma equipe.

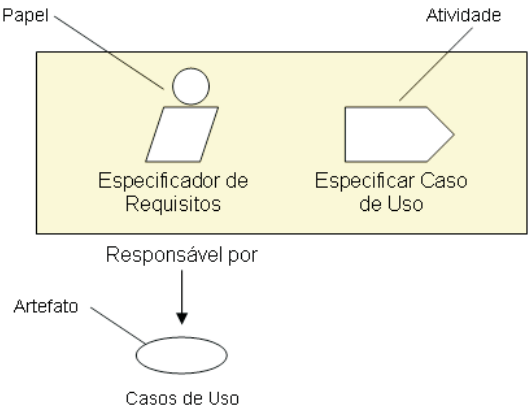

Figura 3.6: Exemplo de um Papel no RUP.

3.2.4 Atividade

O 'Como': São os trabalhos que um indivíduo desempenha dentro de um projeto, podendo criar ou atualizar um artefato. Toda atividade é executada por um papel específico. A unidade de trabalho pode ser hora ou dias. De acordo com a documentação do RUP[1], as atividades são divididas em três etapas:

- **Etapa de Reflexão**: Nesta etapa o papel deve entender o contexto do problema para poder elaborar e examinar os artefatos gerados afim de obter um resultado.

- **Etapa de Execução**: Compreende a criação ou atualização de algum artefato.
- **Etapa de Revisão**: Análise dos resultados comparada com algum parâmetro.

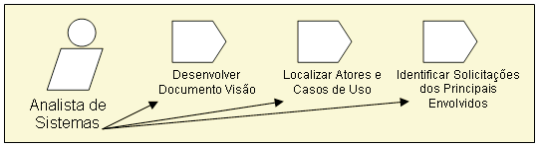

Figura 3.7: Exemplo de Atividades realizadas por um Papel.

3.2.5 Artefato

O 'Que': São os produtos tangíveis obtidos durante o projeto através das atividades realizadas pelos papéis. As atividades utilizam os artefatos como entrada e produzem artefatos de saída. Por exemplo: uma entrevista com o cliente é um artefato de entrada para o artefato de casos de uso.

As informações num projeto fluem através de artefatos. No exemplo da Figura 3.8 as setas mostram como as informações se propagam de um artefato para outro.

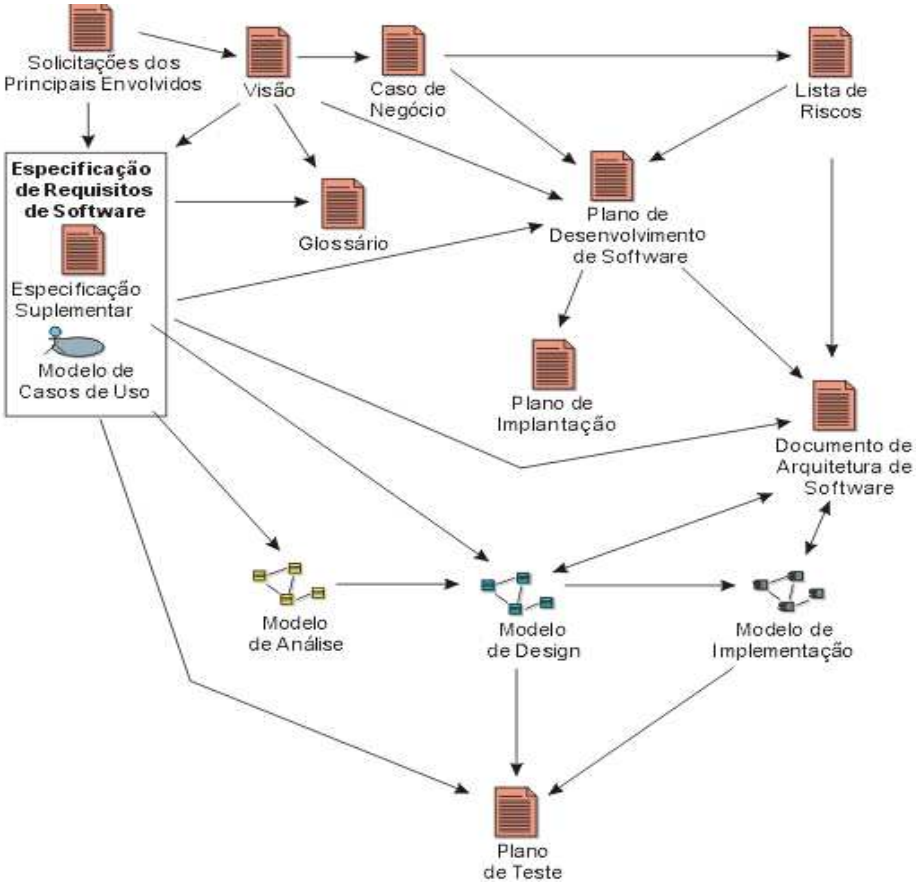

Figura 3.8: Exemplos de artefatos[2].

Capítulo 4

Ciclo de Vida de um Projeto RUP

O ciclo de vida de um projeto RUP propõe uma perspectiva de organização e gerenciamento, onde o processo é dividido em quatro fases distintas, sendo que ao término de cada fase resulta em um marco principal. Um marco é um ponto de decisão, pois ao se concluir uma fase é necessário decidir se deve prosseguir, abortar ou mudar o curso do projeto através da análise do marco. Em resumo, o processo é dividido e organizado em uma sucessão de iterações com objetivos específicos e após a conclusão de um marco, realiza-se uma análise do resultado do marco para tomar uma decisão em relação à próxima fase. Cada fase é um intervalo de tempo entre dois marcos.

A Figura 4.1 representa as quatro fases e o marco ao final de cada uma, tudo em função do

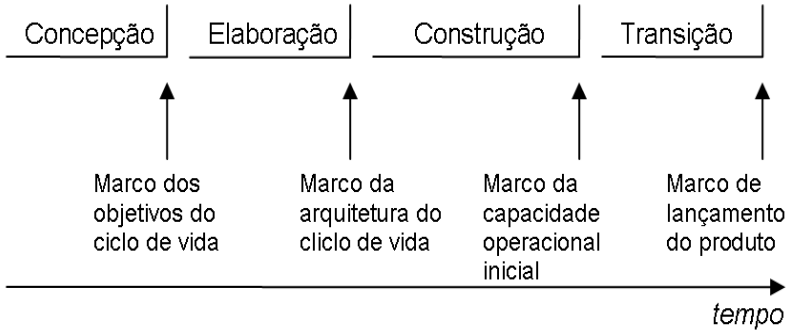

Figura 4.1: Fases e marcos de um projeto.

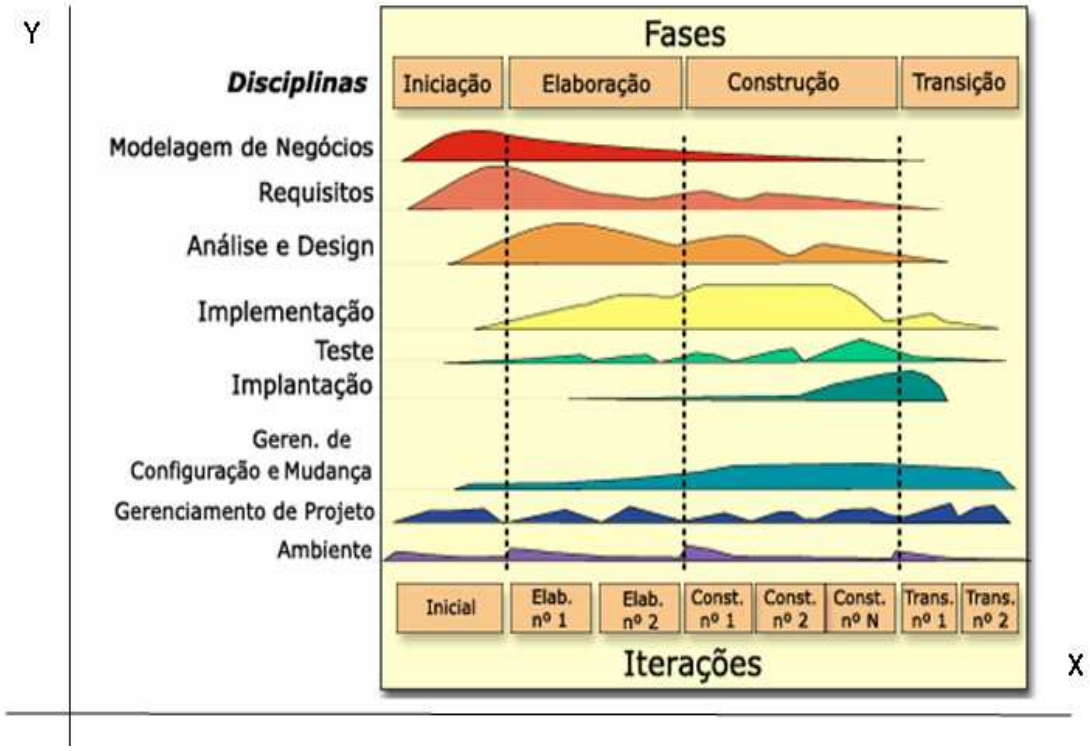

Figura 4.2: Visão Geral do RUP[2].

tempo.

4.1 Fases do RUP

O ciclo de desenvolvimento de um *software* pode ser representado em duas dimensões conforme a Figura 4.2. O eixo x representa a evolução temporal do projeto (aspecto dinâmico) e possui cada uma das quatro fases (Concepção, Elaboração, Construção e Transição) que serão descritas durante este capítulo. O eixo y representa o aspecto estático do projeto, contendo as disciplinas, que representam as atividades e pessoas envolvidas.

4.1.1 Concepção (Iniciação)

Na fase de concepção o objetivo é transformar as necessidades do cliente em um produto, isso acontece com a formalização de um projeto. O projeto deve conter uma visão operacional, custo, riscos e planejamento de seu desenvolvimento. Normalmente a fase de concepção deve durar uma semana dependendo do tamanho do projeto a ser realizado. O ideal é terminar a fase de concepção o mais cedo possível, pois só no final da fase será possível saber a viabilidade em prosseguir com o desenvolvimento. E caso o projeto seja cancelado por motivos financeiros ou não, a equipe teve que dedicar horas de trabalho na fase de concepção.

Atividades básicas

Segundo a documentação do RUP as atividades básicas da fase de concepção são as seguintes:

- **Formular o escopo do projeto**: esta etapa consiste em capturar o contexto, os requisitos e as restrições que envolvem o projeto.
- **Planejar e preparar um caso de negóci**o: envolve o gerenciamento de riscos, a organização da equipe, o plano do projeto, as mudanças de custo, a programação e lucros.
- **Sintetizar uma possível arquitetura**: com a finalidade de ganhar confiança do cliente, pode ser apresentado um modelo ou protótipo do que será o produto.
- **Preparar o ambiente para o projeto**: organização do ambiente de onde ocorrerá o desenvolvimento de *software* e definição dos processos e ferramentas que disponibilizará suporte à equipe de desenvolvimento.

Marco

O marco da concepção é uma avaliação básica do projeto, verificando sua viabilidade e obtendo a concordância dos envolvidos.

Artefatos

A seguir serão relacionados os artefatos básicos da fase de concepção, os modelos serão vistos no capítulo 6 através dos apêndices que descrevem parte do projeto.

- **Documento Visão:** este documento é elaborado durante os primeiros contatos com o cliente. No documento são especificadas de forma geral as necessidades e características fundamentais que o produto deve ter, em relação à visão do cliente. Este documento é de alto nível e deve abranger o sistema como um todo sem entrar em detalhes.

- **Caso de Negócio:** é um plano que foca as características econômicas do projeto com a finalidade de verificar sua viabilidade econômico-financeira.

- **Lista de Riscos:** define as partes mais críticas do projeto e é atualizada sempre quando novos riscos são descobertos ou quando riscos existentes são minimizados ou descobertos.

- **Plano de Desenvolvimento de *Software*:** é um plano que reúne todas as informações do projeto com a finalidade de proporcionar um gerenciamento do mesmo.

- **Glossário:** o glossário traduz todos os termos utilizados pelo cliente e os envolvidos no projeto. É importante disponibilizá-lo na *intranet* da empresa para que os membros da equipe possam estar consultando quando necessário. Imagine por exemplo quando se trata do desenvolvimento de um *software* para a área de saúde, é impossível que os desenvolvedores saibam o significado de todos os termos.

- **Modelo de Caso de Uso:** a partir da UML criam-se modelos que são utilizados como referência durante todo o projeto.

4.1.2 Elaboração

Na fase de elaboração o objetivo é definir a arquitetura do sistema a partir da visão do produto, adquirida na fase de concepção. A arquitetura em determinado momento é definida como a organização ou estrutura de componentes significativos interagindo através de *interfaces*. Para

o desenvolvimento da arquitetura é preciso examinar os requisitos mais significativos, realizar uma avaliação de risco e ter uma compreensão do sistema como um todo.

Atividades básicas

- Ter conhecimento aprofundado dos casos de usos mais significativos, para uma tomada de decisão em relação à arquitetura e o planejamento do sistema.
- Definir uma infra-estrutura, o ambiente de desenvolvimento e as ferramentas a serem utilizadas no projeto.
- Realizar uma seleção e avaliação dos componentes que serão integrados no sistema. Um componente pode ser definido como um módulo, um pacote ou um sistema, com uma função clara e podem ser integrados ao sistema em construção.
- Escalar a equipe do projeto para a fase de construção.
- Realizar as iterações necessárias para estabelecer uma arquitetura, atualizar o plano de projeto e eliminar riscos.
- Analisar os custos iniciais com os apurados no final da fase, para as correções possíveis.

Marco

O marco da fase de elaboração é a definição da arquitetura do projeto, neste momento os objetivos, escopos detalhados, opção de arquitetura e a resolução dos principais riscos serão analisados.

Artefatos

Os artefatos da fase de concepção são refinados, ou seja, são analisados e atualizados, estes procedimentos ocorrerão durante todo o projeto.

- **Lista de Riscos:** a natureza dos novos riscos será da arquitetura, principalmente de requisitos não funcionais, não capturados nos casos de uso do modelo de caso de uso, por exemplo, requisitos legais e de regulamentação.
- **Modelo de Caso de Uso:** nesta fase 80% dos modelos estão completos, todos os atores foram identificados e a maioria das descrições do caso de uso foi desenvolvida.
- **Documento de Arquitetura de *Software*:** o documento é constituído pelas descrições detalhadas dos casos de uso significativos para a arquitetura, identificação dos mecanismos principais e dos elementos do projeto (visão lógica) e da visão da implementação.
- **Modelo de *Design*:** descreve a realização dos casos de uso e serve como uma abstração do modelo de implementação e seu código-fonte.
- **Modelo de Dados:** descreve a representação lógica e física dos dados persistentes no sistema.

4.1.3 Construção

O objetivo da fase de construção é definir algum requisito que falte e desenvolver o sistema. Pode-se dizer que é um processo industrial com ênfase no gerenciamento de recursos, controle de custos, prazos e qualidade. Ocorre a mudança de desenvolvimento intelectual, da fase de concepção e elaboração, para construção de um produto.

Atividades básicas

As atividades básicas da fase de construção são:

- Administrar, controlar recursos e aperfeiçoar o processo.
- Desenvolver completamente os componentes necessários do *software*.
- Implementar e executar os testes para validar a estabilidade do produto.

- Comparar os custos reais com os planejados.
- Avaliar o produto com o documento de visão inicial.

Marco

O marco desta fase consiste em um produto com capacidade operacional inicial, produto beta.

Artefatos

Nesta fase ocorre o refinamento dos artefatos das fases anteriores, com análise e atualizações.

- **O sistema:** trata-se do executável pronto para começar o teste.
- **Plano de Implantação:** descreve o conjunto de tarefas necessárias para instalar o produto desenvolvido de modo que ele possa ser efetivamente transferido para a comunidade de usuários.
- **Conjunto de Teste:** nesta atividade o objetivo é validar a estabilidade do sistema.

4.1.4 Transição

O objetivo da fase de transição é disponibilizar o *software* ao usuário final. É realizado o ajuste final do produto, a configuração, a instalação e os problemas de usabilidade são resolvidos. Todos estes ajustes são realizados com a participação do usuário final, que valida o produto.

Atividades básicas

- Executar a implantação do *software*.
- Finalizar o material de suporte para o usuário final.
- Realizar teste no local de uso do *software*.

- Criar *release*[1] do produto.
- Obter *feedback* do usuário.
- Ajustar o produto com base em *feedback*.
- Disponibilizar o produto para o usuário final.

Marco

O marco desta fase consiste do *release* do produto. A conclusão do projeto é realizada com uma avaliação em termos de satisfação do cliente, se os objetivos iniciais foram alcançados e uma verificação dos custos com o planejado.

Artefatos

- **Produto final:** trata-se do sistema pronto para ser utilizado.
- **Notas de *release*:** registram as mudanças e erros conhecidos em uma versão do *software*.
- **Material de Treinamento:** assegurar que o cliente possa ser auto-suficiente na utilização e manutenção do produto.

4.2 Disciplinas do RUP

O RUP é fortemente centrado no fluxo de trabalho. Os fluxos são divididos em fluxos de engenharia e fluxos de suporte.

Os fluxos ocorrem durante todo o ciclo de vida, variando o nível de intensidade dependendo da fase. A Figura 4.3 representa cada fase e o fluxo de trabalho. Uma iteração se inicia com a modelagem de negócio e termina no fluxo de ambiente. A quantidade de iteração que ocorre

[1] *Release* é uma versão estável e executável do produto. Existem dois tipos de *release*, o interno que é usado apenas pela organização de desenvolvimento, como parte de um marco e o externo que é liberado para usuários finais.

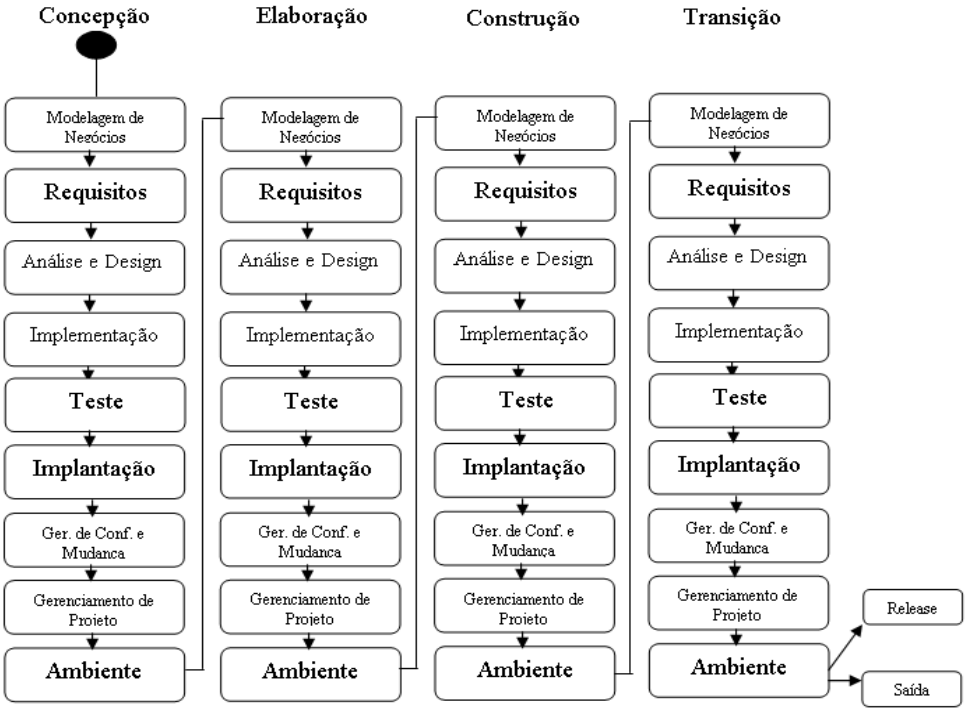

Figura 4.3: Visão Geral das Disciplinas do RUP.

em cada fase, a intensidade com que cada disciplina é executada é especifica de cada projeto. O artefato utilizado para descrever o que acontece em uma iteração é o Plano de Iteração.

4.2.1 Modelagem de Negócios

A disciplina de modelagem de negócios tem como objetivo:

- Entender o funcionamento da empresa para a qual o sistema será desenvolvido.
- Levantar os problemas na empresa e as possíveis soluções.
- Criar uma visão comum da empresa com os envolvidos no projeto, cliente, usuários finais e desenvolvedores.
- Proceder à pesquisa dos requisitos necessários para o projeto.

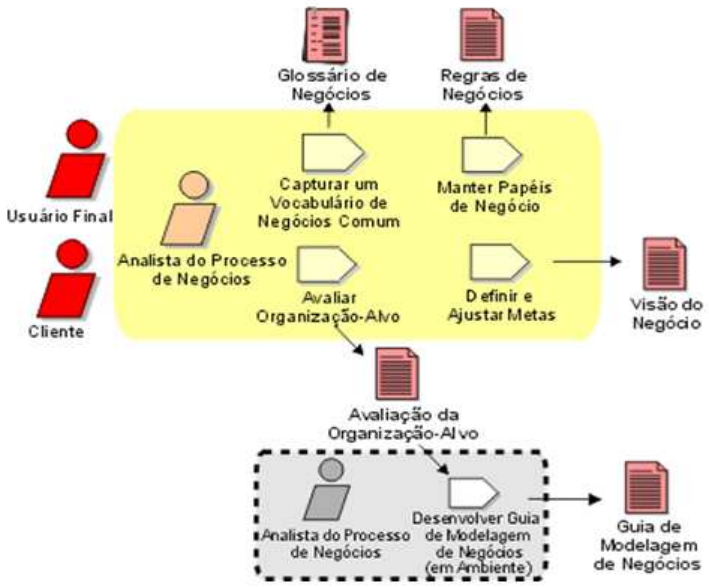

Figura 4.4: Exemplo de fluxo de trabalho na Modelagem de Negócios.

O modelo de negócio compreende a definição do domínio da empresa, cuja abrangência depende do contexto e necessidade, para definir os processos, papéis e responsabilidades da empresa no projeto.

A Figura 4.4 mostra um exemplo de detalhamento do fluxo de modelagem de negócio com os envolvidos no projeto (cliente, usuário final e analista do processo de negócio), os artefatos (glossário de negócios, regras de negócios, visão do negócio, etc) e as atividades (avaliar organização-alvo). O grau de complexidade do detalhamento do fluxo vai depender do projeto a ser desenvolvido.

4.2.2 Requisitos

Requisito é a formalização das necessidades do cliente, que traduz no comportamento do sistema a ser desenvolvido. Esta disciplina visa à identificação dos requisitos de *software* a ser desenvolvido, tendo como objetivo:

- Estabelecer uma conformidade entre o que o sistema deve fazer e as necessidades do

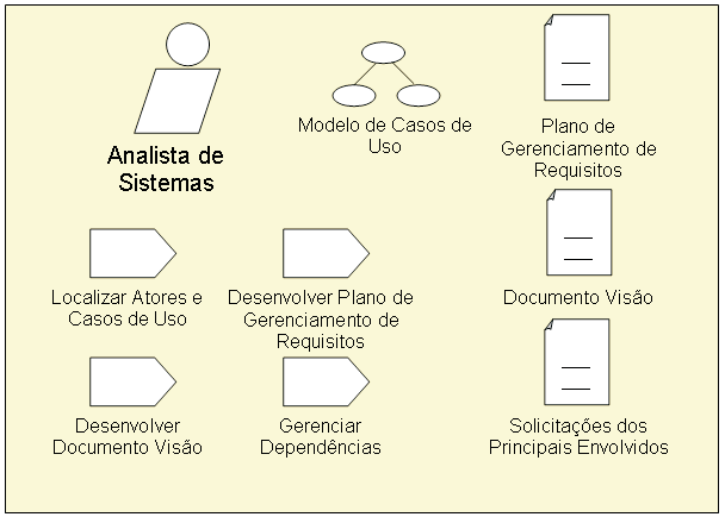

Figura 4.5: Exemplo de atividades e artefatos da disciplina de requisitos.

cliente.

- Fornecer uma visão ampla dos requisitos do sistema.
- Definir a abrangência do que o sistema deve executar.
- Fornecer informações para o planejamento do sistema e das iterações a serem realizadas.
- Subsidiar o levantamento dos custos e tempo necessários para o desenvolvimento do aplicativo.
- Definir um protótipo com as *interfaces* para o usuário.

Os artefatos de entrada desta disciplina são os artefatos gerados na disciplina de modelagem de negócios. A Figura 4.5 apresenta algumas atividades e artefatos da disciplina de requisitos.

4.2.3 Análise e *Design*

A fase de análise e *design* fornece uma preparação para a codificação, ou seja, é a entrada para a implementação. A finalidade desta disciplina é:

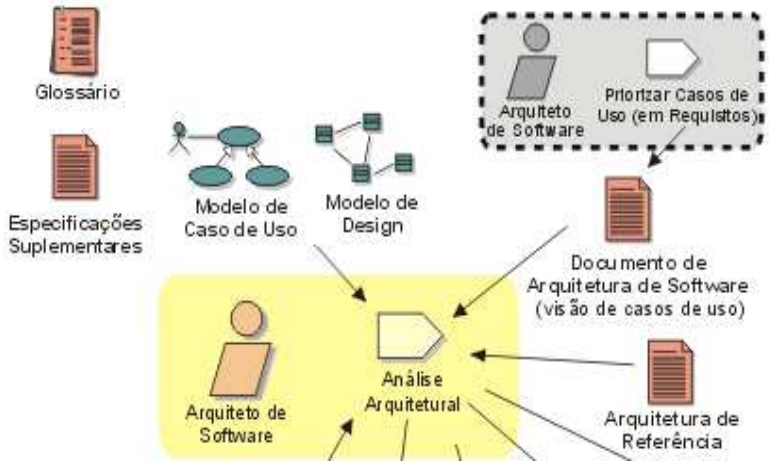

Figura 4.6: Exemplo de atividades típicas da Análise e *Design*.

- Conversão dos requisitos em um projeto do sistema a ser desenvolvido. Por exemplo: transformar os requisitos em casos de uso e estes em classes.

- Adaptar o projeto ao ambiente de implementação. Um exemplo seria utilizar o paradigma de orientação a objeto tanto no projeto quanto na implementação.

- Desenvolver uma arquitetura para o sistema. Segundo a documentação do RUP a arquitetura compreende a organização e a estrutura de controle: protocolos de comunicação, sincronização e acesso a dados; atribuição de funcionalidade a elementos de *design*; distribuição física; composição de elementos de *design*; escalonamento e desempenho; e seleção entre as alternativas de *design*.

A finalidade da Figura 4.6 é apresentar uma visão global da disciplina em foco, o envolvido é o arquiteto de *software*. Os artefatos de entrada são os gerados nas disciplinas anteriores e os artefatos como documento de arquitetura de *software*, guia de *design* são à saída da disciplina.

4.2.4 Implementação

A implementação na engenharia de *software* é uma atividade que compreende a codificação propriamente dita. Tendo como finalidade:

- Definir classes e objetos em termos de componentes, já que o paradigma utilizado no projeto de desenvolvimento é a orientação a objeto.
- Planejar o desenvolvimento do código em termos de subsistema, com a finalidade de facilitar a codificação e os testes.
- Programar os testes nos componentes desenvolvidos como unidades.
- Realizar a integração dos subsistemas desenvolvidos pelos implementadores individuais.

A implementação ocorre na construção, que é uma versão operacional de um sistema ou parte de um sistema que demonstra um subconjunto das capacidades fornecidas no produto final, a integração que faz a combinação dos componentes separados gerando um sistema completo. A integração é incremental, significa que o código é escrito e testado por partes pequenas, para depois formar o todo.

Nesta disciplina os protótipos são usados para reduzir os riscos do projeto. A Figura 4.7 apresenta o detalhamento da disciplina de implementação que tem como objetivo organizar o desenvolvimento dos componentes e o processo de construção para proporcionar a menor quantidade de conflitos.

4.2.5 Teste

Na disciplina de teste a idéia principal é localizar e expor os pontos fracos do produto, enfatizando as deficiências do mesmo. Ao mesmo tempo é realizada uma avaliação da qualidade do produto, através de várias práticas. A avaliação da qualidade visa garantir que os padrões do projeto estejam corretos e sejam seguidos por toda a equipe do projeto.

As práticas centrais da disciplina de teste são:

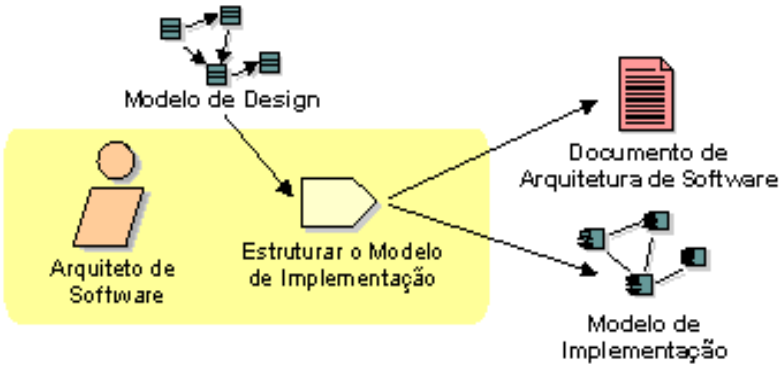

Figura 4.7: Exemplo de fluxo de trabalho da disciplina de Implementação.

- Realizar a localização e documentar os principais defeitos na qualidade do *software*.
- Fazer avaliações constantes sobre a conformidade entre os padrões de projeto estipulados pela empresa de desenvolvimento e o que foi realmente construído.
- Validar as funções do *software* conforme foi projetado.
- Verificar se os requisitos foram implementados de forma adequada.
- Realizar um levantamento das incertezas que envolvem as demais disciplinas.

A Figura 4.8 mostra um exemplo de detalhamento do fluxo de trabalho de teste, envolvendo artefatos e atividades que identificam os objetivos e os produtos liberados do esforço de testes, define também o método que será utilizado. O papel de conduzir os testes é desempenhado pelo gerente de testes auxiliado pelo analista de testes.

4.2.6 Implantação

A disciplina de implantação compreende toda a sistemática que garante que o *software* desenvolvido esteja à disposição do usuário final. A implantação do produto final ocorre de forma compacta, personalizada ou através da disponibilidade do produto pela internet. Em cada uma destas formas de instalação o produto é testado no local da implantação.

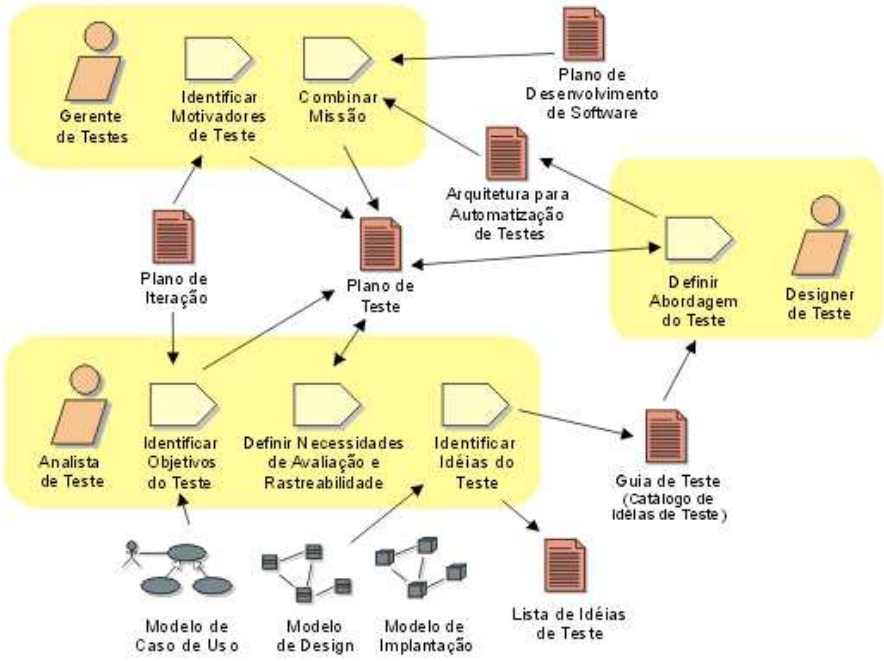

Figura 4.8: Exemplo de fluxo de trabalho da disciplina teste.

Nesta disciplina há o desenvolvimento do plano de implantação que fornece uma agenda detalhada dos processos, pessoas responsáveis e as garantias necessárias para o sucesso da implantação do produto, no local estipulado pelo cliente. A Figura 4.9 representa os passos, em termos de envolvidos, atividades e artefatos, necessários para a instalação de uma unidade de implantação. A finalidade de criação da unidade de implantação é testar o produto antes da entrega definitiva ao cliente.

4.2.7 Gerenciamento de Configuração e de Mudança

A disciplina de gerenciamento de configuração (CM) e gerenciamento de mudança (CRM) tem como objetivo localizar e registrar o ciclo de vida de desenvolvimento, através dos artefatos. Isto é realizado em temos da evolução das diversas etapas do projeto. Provendo o controle dos diversos artefatos criados, pela equipe de desenvolvimento, e consolidando sua evolução.

Uma forma clássica de representar as funções da disciplina desse item consiste em utilizar o cubo CCM, Modelo de Maturidade de Capacidade (CMM) do *Software Engineering Institute*

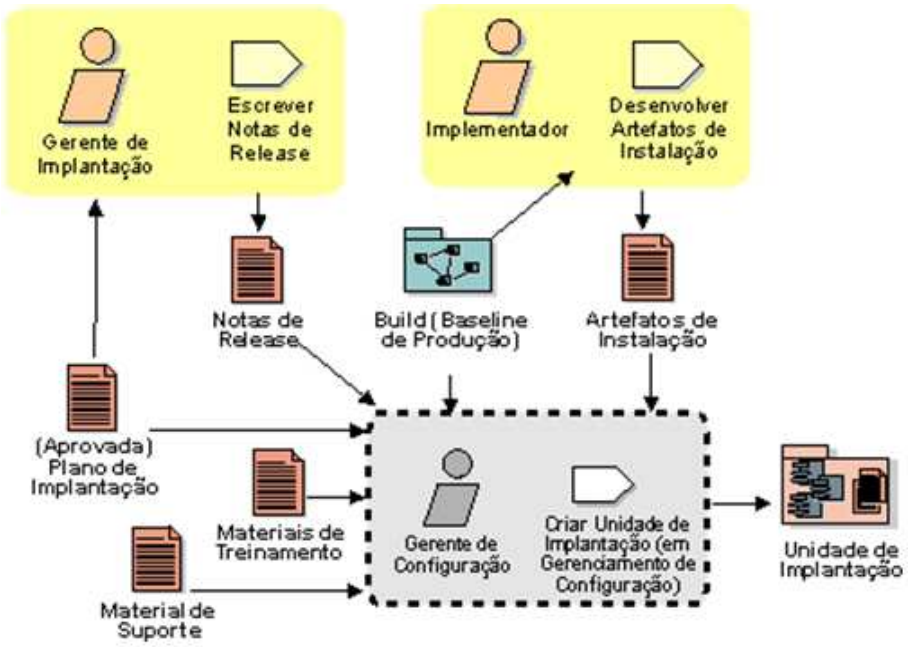

Figura 4.9: Exemplo de fluxo trabalho da disciplina de Implantação.

(SEI), Figura 4.10. O cubo tem três fases e cada uma representa uma função que está inter-relacionada.

- A face do gerenciamento de configuração (CM) está ligada a estrutura do *software*, representa as versões, os artefatos e as configurações, que são conjuntos de artefatos relacionados.

- A segunda fase é o gerenciamento de solicitações de mudança (CRM) que consiste nas solicitações de alguma alteração de itens do projeto, realizadas por clientes internos ou externos. É analisado o impacto que as mudanças provocarão no projeto em termos de custo e tempo. As mudanças estão relacionadas à estrutura de processo.

- A última face do estado e medida diz respeito à estrutura de controle do projeto, o que foi terminado e o que falta ser feito, problemas que exigem atenção.

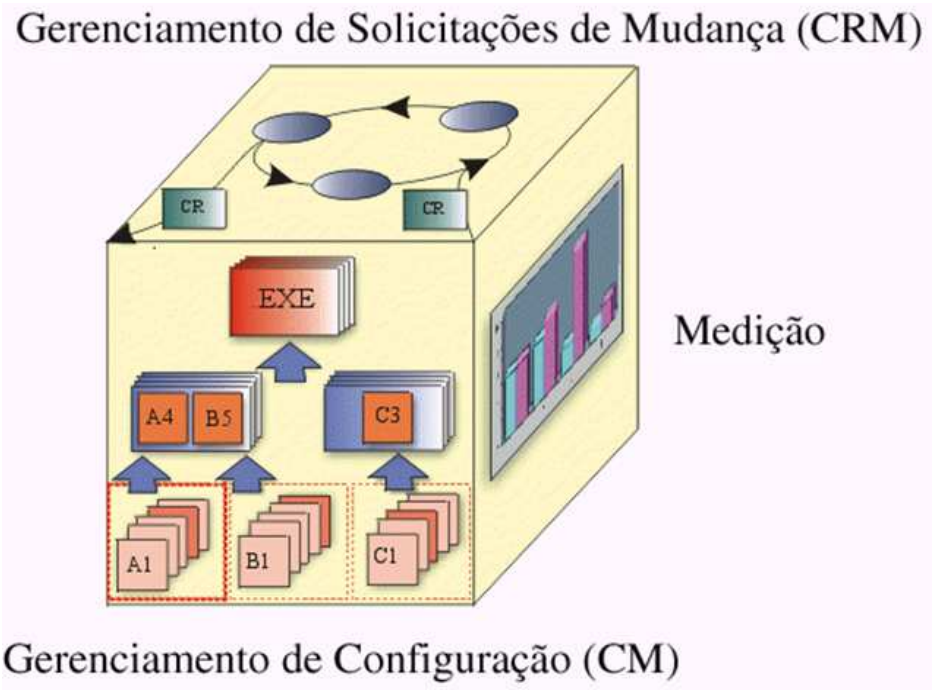

Figura 4.10: Cubo CCM.

4.2.8 Gerenciamento de Projeto

O gerenciamento de projeto envolve toda a parte de orçamento, montagem da equipe, administração de riscos, geração de um plano de iterações e outros elementos importantes para atender as necessidades do cliente. O gerenciamento de projeto tem como finalidade:

- Avaliar o escopo e riscos do projeto.
- Monitorar e controlar as várias etapas do projeto
- Planejar e avaliar cada iteração e fase.
- Funcionar como um instrumento para acompanhamento do projeto.

A Figura 4.11 demonstra um exemplo do fluxo de atividades da disciplina de Gerenciamento de Projeto, onde as atividades são conduzidas pelo gerente de projeto.

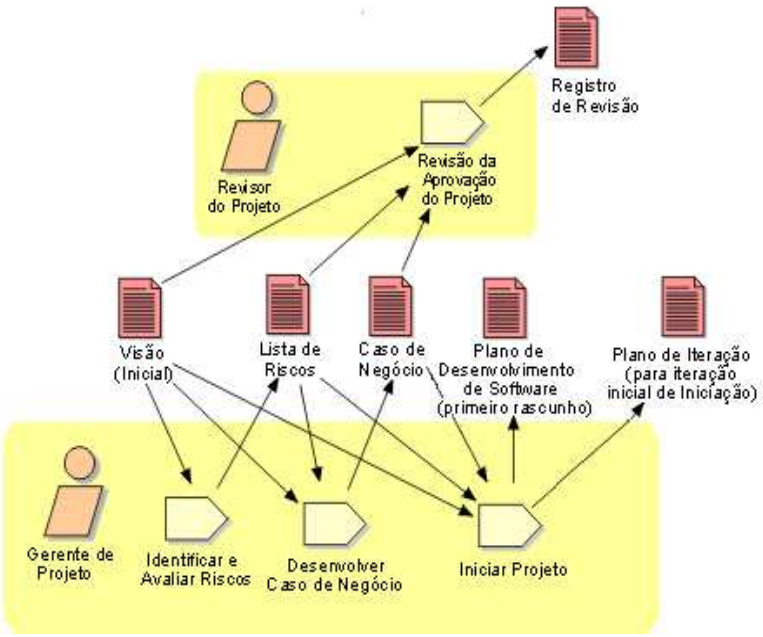

Figura 4.11: Exemplo de fluxo da disciplina de Gerenciamento de Projeto.

4.2.9 Ambiente

A meta final da disciplina de ambiente é fornecer suporte ao projeto em termos de processo, métodos e ferramentas, que inclue:

- Seleção e aquisição de ferramentas.
- Configuração do processo antes de iniciar o projeto.
- Melhoria do processo para o trabalho de desenvolvimento.

A Figura 4.12 mostra um exemplo de ambiente para o desenvolvimento incluindo as atividades, artefatos, envolvidos e as ferramentas necessárias.

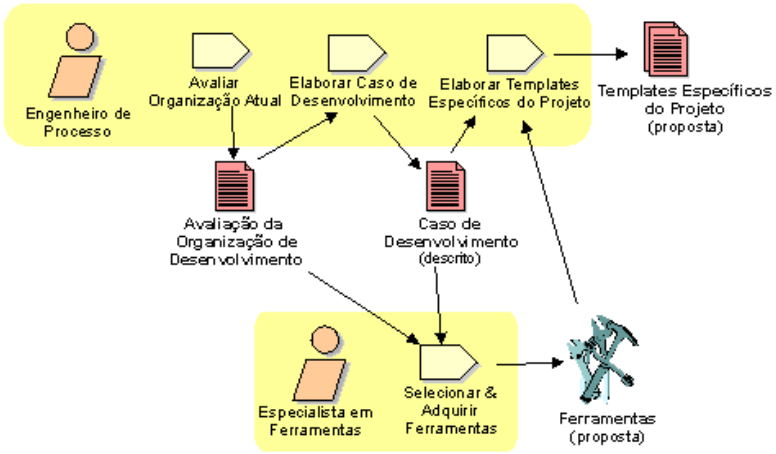

Figura 4.12: Exemplo de fluxo da disciplina de Ambiente.

Capítulo 5

A Importância da Modelagem para o Desenvolvimento de *Software*

Um exemplo de modelagem de sucesso ocorre na construção civil. O desejo de construir uma casa leva um indivíduo a procurar um engenheiro civil. A partir das informações fornecidas pelo cliente tais como: quantidade de quartos, salas, o melhor local para a cozinha, um estudo das condições do terreno e o local onde será realizada a construção e etc. Com base nestas informações o engenheiro em conjunto com um arquiteto começam a elaborar um projeto.

Na elaboração do projeto da construção da casa é gerada uma planta arquitetônica, que mostra onde deve ficar as paredes, os movéis e o tipo de acabamento que será aplicado. São também elaboradas as plantas do sistema elétrico, hidráulico e estrutural, ou seja, tudo é pensado antes.

O projeto da casa constitui um modelo que representa as diversas abstrações, que formam uma visão global da casa. O projeto (modelo) é apresentado ao cliente com a finalidade de aprovação, ou seja, se o que está na planta do projeto corresponde as especificações do cliente. Caso o cliente solicite alguma alteração, o arquiteto faz as modificações necessárias e apresenta o projeto novamente ao cliente para validação. Até neste momento nenhum tijolo ainda é assentado, mas já é definido o tempo que a construção da casa irá consumir e o orçamento.

A próxima etapa é a construção da casa de acordo com o projeto (modelo visual) validada

pelo cliente. A próxima etapa consiste em apresentar o projeto ao encarregado da construção da casa, mestre de obra, pedreiros, serventes, encanador, pintor etc. Todas as pessoas que participarão da construção terão uma visão da sua parte na obra através da planta.

A última etapa, que é a entrega do produto final dentro do prazo estipulado, ou seja, a casa já pronta para ser ocupada atendendo todas as expectativas do cliente e de acordo com o modelo criado pelo engenheiro e arquiteto.

Conforme pode ser observado nos parágrafos anteriores, foi feito um breve resumo das fases de uma construção civil, onde o principal objetivo foi dar ênfase na modelagem da casa, que foi visto como uma simplificação da realidade, ou seja, através modelo apresentado (planta da casa) o cliente pode ter uma visão da casa, satisfazendo suas expectativas.

No contexto de *software*, os modelos visuais fazem à mesma coisa e são chamados de modelagem visual ou modelagem de sistemas. É como as plantas do sistema. Através das plantas é possível planejar os cômodos antes de construí-los. Um modelo visual ajuda a planejar o sistema antes de construí-lo servindo como um guia a ser seguido em uma fase de construção de *software*.

Constrói-se modelos para compreender melhor o sistema que está sendo desenvolvido. Booch, Rambaugh e Jacobson [7]. Com a modelagem, é possível alcançar quatro objetivos:

1. Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja.
2. Os modelos permitem especificar a estrutura ou o comportamento de um sistema.
3. Os modelos proporcionam um guia para a construção do sistema.
4. Os modelos documentam as decisões tomadas.

A modelagem não se restringe apenas no desenvolvimento de grandes *softwares*. Até mesmo sistemas mais simples podem utilizar os benefícios da modelagem. Dependendo da complexidade do sistema a ser desenvolvido fica impossível compreendê-lo sem uma correta modelagem. O mesmo vale para a construção civil, por menor que seja uma casa e por mais complexo que seja a construção de um enorme prédio, ambos precisam ser planejados e modelados.

5.1 *UNIFIED MODELING LANGUAGE* (UML) como Parte Integrante do RUP

A UML é o braço direito do RUP, mas isso não significa que ela seja dependente do RUP, na verdade ocorre o contrário, pois a UML é totalmente independente de processo ou modelo, podendo ser utilizada para diversas finalidades. O RUP apenas utiliza intensivamente os recursos da UML para a geração de seus artefatos referentes a um *software*.

A UML é uma linguagem padrão para a elaboração da estrutura de projetos de *software* podendo ser empregada para a visualização, especificação, construção e a documentação de artefatos que façam uso de sistemas complexos de *software* [7].

Desta forma a UML é utilizada para a modelagem de sistemas abrangendo todas as visões necessárias relacionadas ao desenvolvimento e implantação de um sistema. A UML atualmente é composta por treze diagramas onde cada um possui sua própria semântica, sendo que, cada diagrama é utilizado conforme as fases de um projeto de *software*. Isso significa que, em uma fase inicial (fase de Concepção do RUP, por exemplo) de projeto é utilizado um determinado diagrama que ao mesmo tempo se torna requisito fundamental para o uso do próximo diagrama e assim sucessivamente.

Da mesma forma que a engenharia civil utiliza na elaboração de um modelo vários projetos (elétrica, hidráulica, estrutural e arquitetônico, etc) para a construção de um imóvel, a engenharia de *software* possui a sua disposição os treze diagramas da UML com a finalidade de oferecer uma visão geral. Cada diagrama representa um enfoque diferente do sistema ou parte dele, analisando sobre uma determinada visão.

A UML é uma linguagem de modelagem visual, pois facilita um melhor entendimento e absorção de idéias, não só pelo desenvolvedor, mais principalmente pelo usuário.

No exemplo a seguir, é mostrado a descrição de uma classe, em seguida a representação gráfica da mesma classe.

- Exemplo Descritivo de Classe: Classe fornecedor possui os atributos: Razão Social

Figura 5.1: Exemplo de Classe.

(*String*), CNPJ (*String*), Contato (*String*).

- Exemplo Visual da Classe: A Figura 5.1 mostra a classe de forma visual.

A representação gráfica é mais simples e facilita o entendimento, que uma descrição extensa.

5.2 DIAGRAMAS DA UML 2.0

Um diagrama pode ser definido como uma apresentação gráfica de um conjunto de elementos relacionados entre si, que possibilita diferentes visões de um sistema.

Conforme mencionado anteriormente, a UML define treze tipos de diagramas, que são divididos em duas categorias: diagramas estruturais ou estáticos e diagramas dinâmicos, conforme pode ser visto a seguir:

Diagramas Estáticos

- Diagrama de Classes
- Diagrama de Objeto
- Diagrama de Componentes
- Diagrama de Pacotes

- Diagrama de Implantação
- Diagrama de Estrutura Composta

Diagramas Dinâmicos

- Diagrama de Caso de Uso
- Diagrama de Visão Geral
- Diagrama de Seqüência
- Diagrama Temporal
- Diagrama de Comunicação
- Diagrama de Atividades
- Diagrama de Máquina de Estados

Nos diagramas estáticos, as suas características não sofrem alterações no decorrer do tempo. Os diagramas estáticos podem ser associados ao modelo de estrutura e o modelo de arquitetura. O modelo de Estrutura é utilizado para visualizar, especificar, construir e documentar as partes internas do sistema, como as classes, objetos, interfaces e seus relacionamentos e modelo lógico do banco de dados. O modelo de Arquitetura tem como finalidade especificar as partes físicas e externas do sistema, que são os componentes, nós, programas fontes, modelo físico do banco de dados, dentre outros.

Em contrapartida os diagramas tidos como dinâmicos mostram como o sistema responde às requisições ou evoluções durante a variável tempo. Os diagramas dinâmicos estão associados ao modelo de comportamento que é utilizado para troca de mensagens entre os objetos e componentes no tempo, bem como a mobilidade de componentes entre os nós da rede.

5.2.1 Diagrama de Casos de Uso

Antes de entender o que é um caso de uso, é necessário entender o conceito de cenário. Cenário é uma seqüência de passos que descreve uma interação entre um usuário e um sistema. A partir desse conceito o caso de uso é definido como um conjunto de cenários ligados por um objetivo comum de um ator. Um caso de uso é uma representação de várias ações para a realização de uma atividade. Há necessidade das ações terem uma afinidade entre si para haver um caso de uso. Um caso de uso é utilizado para capturar os requisitos funcionais do sistema.

Normalmente quando modelamos sistemas, definimos um caso de uso principal e outros que serão os alternativos. O caso de uso apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma idéia geral de como o sistema irá se comportar.

Os critérios para o desenvolvimento dos casos de uso segundo Guedes[8] são os seguintes:

- Descrever o que o sistema faz.
- Descrever o comportamento do sistema com base na visão dos atores.
- Cada caso de uso deve produzir algo de valor para um ou mais atores.
- Cada caso de uso deve abordar um processo completo do negócio.

Um diagrama de Caso de Uso é composto por:

- **Ator**: É um papel que um usuário desempenha em relação ao sistema. O nome do ator deve ser um sujeito. Podem ser usuários ou máquina e todos devem ser descritos com detalhes. O ator é representado pelo símbolo de um "boneco magro".
- **Linha de Comunicação**: Não é necessário identificar a comunicação entre atores e casos de uso, mas é uma boa prática.
- **Casos de Uso**: Referem-se aos serviços, tarefas ou funções que podem ser utilizados de alguma maneira pelos usuários, ou seja, são utilizados para expressar e documentar os comportamentos pretendidos para as funções do sistema [8]. O nome do caso de uso deve

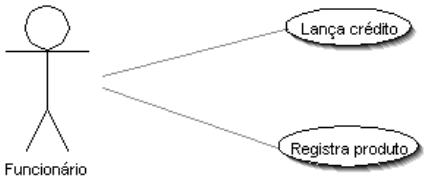

Figura 5.2: Exemplo de Caso de Uso.

ser um verbo que facilite a identificação da funcionalidade do mesmo. O caso de uso é representado por uma elipse conforme a Figura 5.2.

Em um diagrama de caso de uso, existe também os tipos de relacionamentos conforme mostrados a seguir:

- **Associação**: Representam as interações ou relacionamento entre os atores, e caso de uso e entre os casos de uso e outros casos de uso, ou seja, a comunicação entre atores e casos de uso, por meio do envio e recebimento de mensagens.
- **Especialização/Generalização**: Ocorre entre caso de uso ou entre atores quando apresentam características semelhantes, possuindo pequenas diferenças. Em outras palavras ocorre quando um ator ou caso de uso herda características de outro.
- **Extensão(*Extend*)**: É utilizado para descrever cenários opcionais de um caso de uso. Indica que um caso de uso terá seu procedimento acrescido de outro caso de uso.
- **Inclusão(*Include*)**: É quando existe um serviço, situação ou rotina comum a mais de um caso de uso.

5.2.2 Diagrama de Classes

O diagrama de classes permite a visualização das classes que compõe o sistema, com seus nomes, atributos e operações, bem como o relacionamento, complementação e transmissão de informações entre si. Representa as estruturas estáticas do sistema, sendo compostas pelas

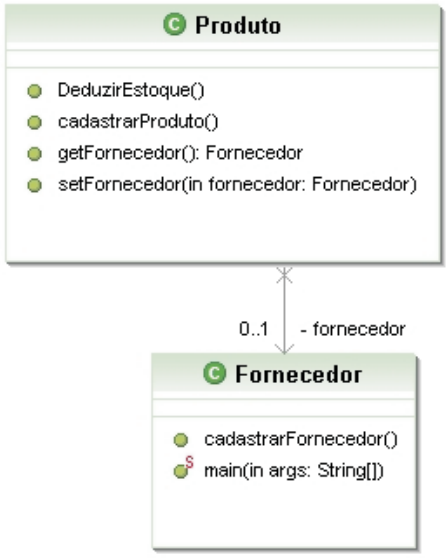

Figura 5.3: Exemplo de Relacionamento entre Classes.

classes de negócios, classes de interface com o usuário e com outros sistemas, e as classes de controle, responsáveis pelo controle das transações.

Em UML as classes são representadas por um retângulo dividido em três compartimentos contendo:

- **Nome**: que conterá apenas o nome da classe modelada.
- **Atributos**: que possuirá a relação de atributos que a classe possui em sua estrutura interna.
- **Operações**: que serão os métodos de manipulação de dados e de comunicação de uma classe com outras do sistema.

5.2.3 Diagrama de Objetos

O diagrama de objeto é um complemento do diagrama de classe. Seu objetivo é fornecer uma visão dos valores armazenados pelos objetos das classes, em um determinado momento da

Figura 5.4: Exemplo de Diagrama de Objetos.

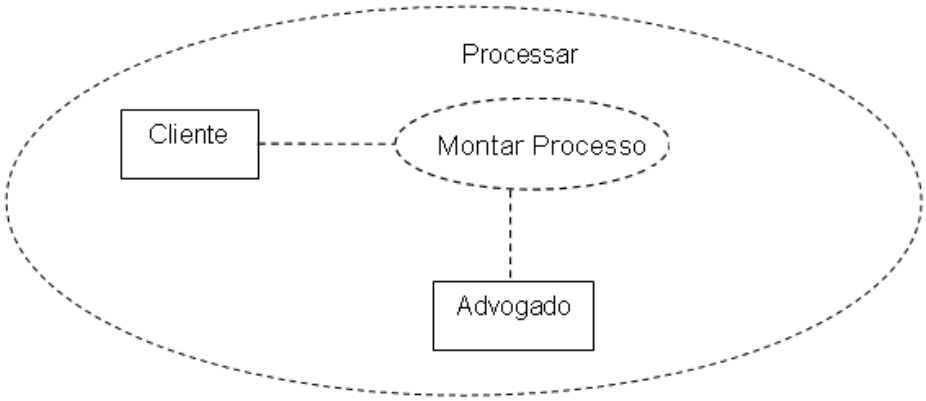

Figura 5.5: Exemplo de Diagrama de Estrutura Composta.

execução de um processo. Em UML um objeto é mostrado como uma classe só que seu nome é sublinhado, e o nome do objeto pode ser mostrado opcionalmente precedido do nome da classe. Veja Figura 5.4.

5.2.4 Diagrama de Estrutura Composta

Representa um momento de colaboração entre interfaces, objetos ou classes. Descreve a estrutura interna de um classificador, como uma classe ou componente, detalhando as partes internas que o compõem e a forma como estas se comunicam e colaboram entre si. A Figura 5.5 representa o diagrama de estrutura composta.

5.2.5 Diagrama de Seqüência

Preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Determina quais condições devem ser satisfeitas e quais métodos devem ser disparados entre os objetos e a ordem em que os eventos ocorrem.

O diagrama de seqüência baseia-se no diagrama de caso de uso, permitindo documentar o caso de uso. Segundo Medeiros[9], o diagrama de sequência(Figura 5.6) melhora o diagrama de classes permitindo que se acrescente ou retire métodos e/ou atributos desenecessários de um conjunto de classes. O diagrama é composto por:

- **Ator**: Possui o mesmo significado e representação que nos diagramas de caso de uso.
- **Objetos**: É uma instância das classes envolvidas no processo. É representado por uma caixa na parte superior de uma linha tracejada verticalmente. A linha vertical é chamada de linha da vida do objeto, e representa a vida do objeto durante a interação.
- **Mensagem**: É representada por uma flecha entre as linhas de vida de dois objetos. Cada mensagem deve ter um nome, é comum incluir os argumentos e algumas informações de controle.

5.2.6 Diagrama de Comunicação

Concentra-se em como os objetos estão vinculados e quais mensagens são trocadas entre si durante o processo. Os objetos são distribuídos de forma similar ao diagrama de seqüência obedecendo à seqüência de mensagens. As comunicações entre os objetos são representadas por uma ligação simples e possui uma numeração seqüencial, sendo acompanhada de outras informações como condições e iterações. Figura 5.7.

5.2.7 Diagrama de Estado

O Diagrama de Estado procura acompanhar as mudanças sofridas por um objeto dentro de um determinado processo, podendo ser utilizado para representar os estados de um caso de uso ou

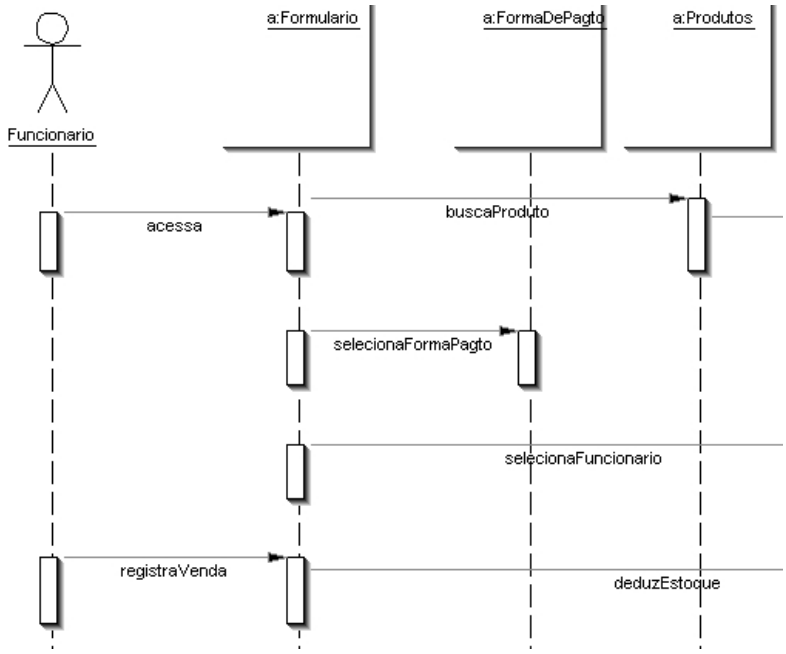

Figura 5.6: Exemplo de Diagrama de Seqüência.

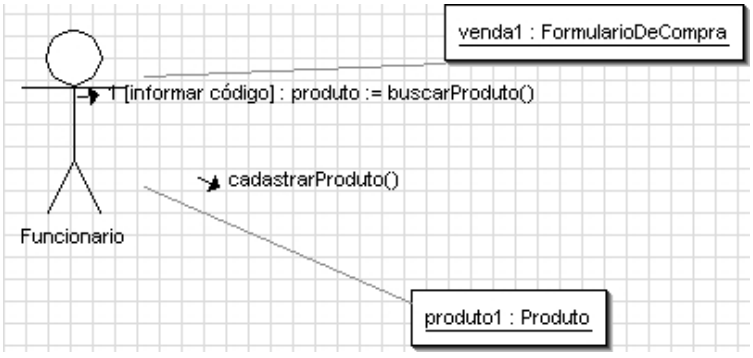

Figura 5.7: Exemplo de Diagrama de Comunicação.

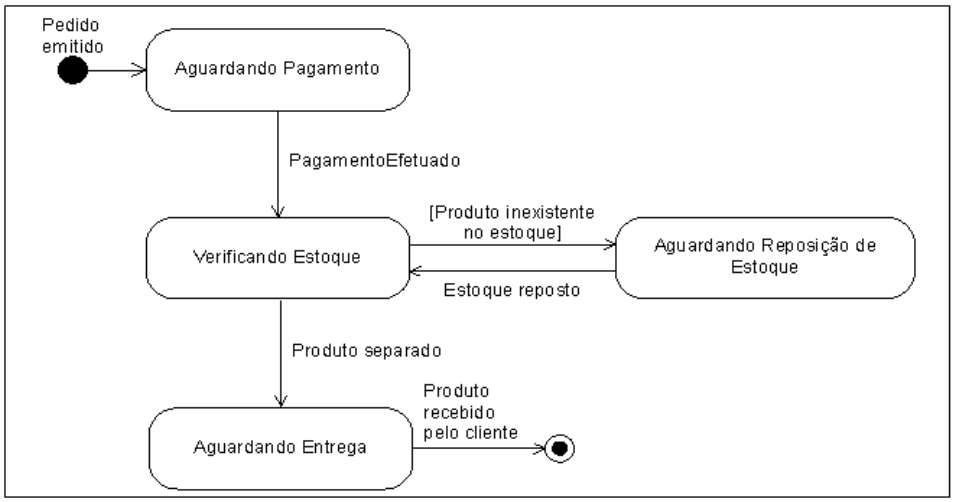

Figura 5.8: Exemplo de Diagrama de Estado.

mesmo os estados gerais de um sub-sistema ou de um sistema completo. Em outras palavras o diagrama de máquina de estados descreve o comportamento de objetos por meio de seqüências de estados e ações que ocorrem durante a sua vida. Figura 5.8.

5.2.8 Diagrama de Atividades

Preocupa-se em descrever os passos a serem percorridos para a conclusão de uma atividade especifica, concentra-se na representação do fluxo de controle de uma atividade. Figura 5.9.

5.2.9 Diagrama de Componentes

Esse diagrama representa os componentes do sistema e sua associação com outros componentes, por exemplo, em um sistema pode-se ter vários arquivos com extensão .java, cada arquivo desse é considerado um componente [10]. Vários componentes podem formar um módulo completo, esse módulo é também chamado de componente que pode se associar com outro. Figura 5.10.

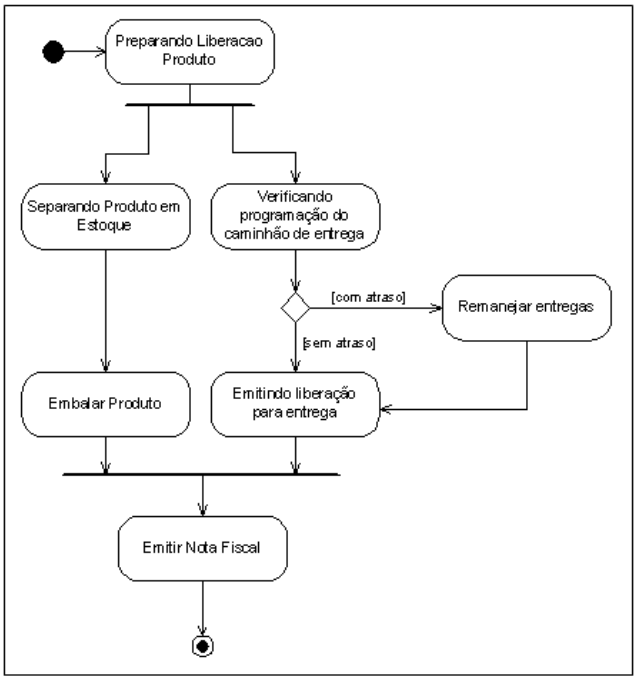

Figura 5.9: Exemplo de Diagrama de Atividades.

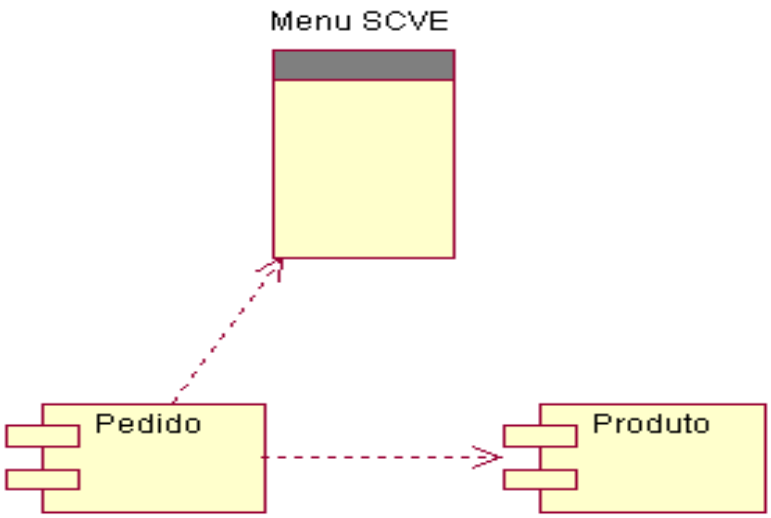

Figura 5.10: Exemplo de Diagrama de Componentes.

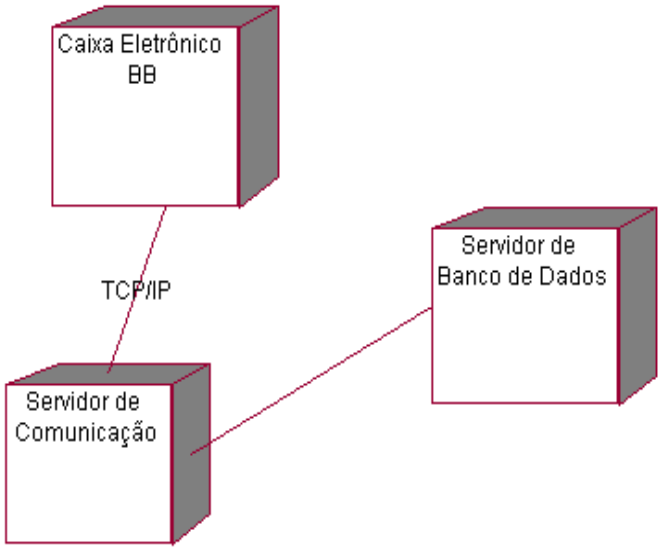

Figura 5.11: Exemplo de Diagrama de Implantação.

5.2.10 Diagrama de Implantação

Determina as característica de como será a implantação do sistema através de características físicas (*hardware*), protocolos de comunicação, tipos de servidores e etc. A Figura 5.11 exemplifica de forma bem direta o diagrama de implantação.

5.2.11 Diagrama de Pacotes

Tem por objetivo representar os subsistemas englobados por um sistema de forma a determinar as partes que o compõem. É utilizado para organizar elementos de modelo e mostrar dependências entre eles. Figura 5.12.

5.2.12 Diagrama de Interação - Visão Geral

O diagrama de interação é uma especialização do diagrama de atividades representando interações, promovendo uma visão geral do fluxo de controle. No diagrama de visão geral existem dois tipos de quadros: quadros de ocorrência interação, que normalmente fazem referência a

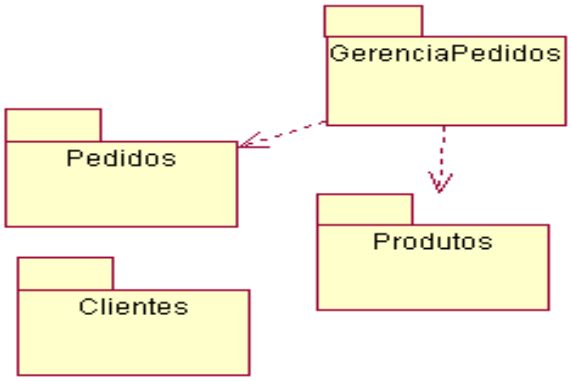

Figura 5.12: Exemplo de Diagrama de Pacotes.

um diagrama de interação e quadro de interação que contém qualquer tipo de diagrama de interação. Figura 5.13.

5.2.13 Diagrama de Tempo

Descreve a mudança no estado ou condição de uma instância de uma classe ou seu papel durante um tempo, ou seja, enfoca as mudanças de estado de um objeto ao longo do tempo. Figura 5.14.

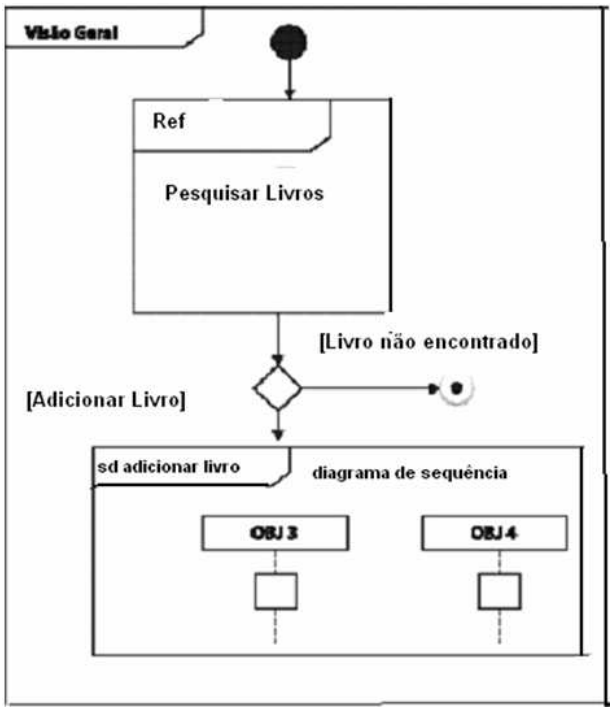

Figura 5.13: Exemplo de Diagrama de Interação.

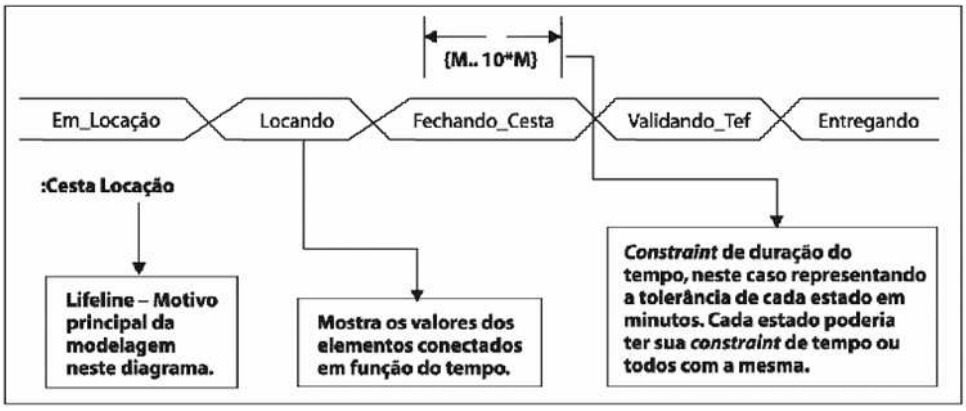

Figura 5.14: Exemplo de Diagrama de Tempo[9].

Capítulo 6

Aplicando a Metodologia RUP para o Desenvolvimento de um Projeto de *Software*

Para demonstrar a utilização da metodologia RUP, foi proposto o desenvolvimento de um projeto de *software* que será chamado de Sistema de Controle de Vendas e Estoque (SCVE). O sistema irá atender algumas necessidades específicas de uma empresa real, cujo nome fictício da empresa será Locadora de Games, onde serão elaborados todos os artefatos necessários para que o projeto futuramente possa ser codificado durante a fase de construção e posteriormente passar pela fase de transição, onde ocorrerá a implantação. Este é um projeto de desenvolvimento de *software* que irá percorrer apenas pela fase de concepção e elaboração, ou seja, após o término destas duas fases conclui-se parte do projeto de *software*. No entanto, foi feito o suficiente para demonstrar o projeto de *software* baseado no RUP.

É importante que o leitor tenha em mente que, o projeto de *software* não se conclui em 100% na fase de elaboração, pois 50% do esforços ainda são desempenhados na fase de construção e 10% durante a fase de transição conforme mostrado na Figura 6.1.

Nos próximos tópicos serão apresentados todos os artefatos criados durante a fase de con-

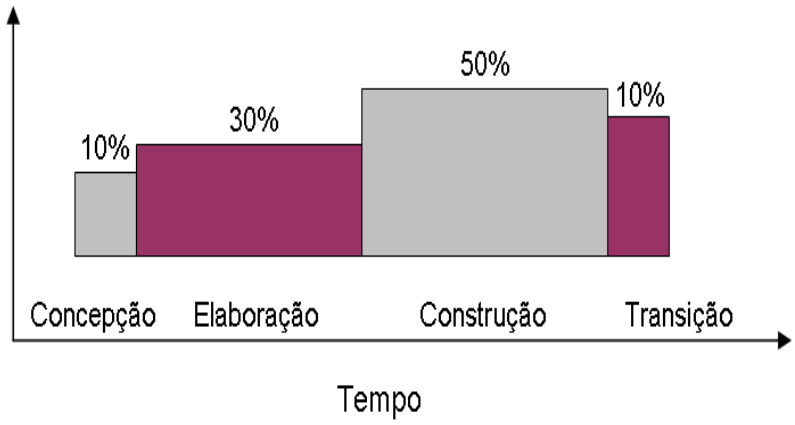

Figura 6.1: Perfil de um projeto típico[2].

cepção e elaboração.

6.1 Fase de Concepção do Projeto SCVE

O passo inicial na fase de concepção (início do projeto) foi aplicar parte dos esforços na disciplina de requisitos para identificar a visão dos envolvidos em relação ao produto a ser desenvolvido (SCVE). Esta etapa resultou no artefato visão (documento visão) que fornece uma base de alto nível para que o cliente possa aprovar as funcionalidades do sistema de acordo com suas necessidades. Ao mesmo tempo, o artefato visão servirá como entrada na elaboração de novos artefatos.

Procurou-se abordar o sistema da forma mais simples o possível, não levando em consideração a variável custo, por se tratar do primeiro projeto da equipe baseado no RUP.

Todos os artefatos foram construídos com base nos *templates*[1] disponíveis na documentação do RUP. No entanto não é necessário seguir a risca os *templates*, pois os mesmos podem ser customizados de acordo com a necessidade do projeto.

A partir deste ponto, serão mencionados todos os artefatos criados durante a fase de concepção. É aconselhável que o leitor acompanhe os apêndices conforme os artefatos vão sendo

[1] *Templates* são Modelos de Documentos

mencionados a seguir:

- Documento Visão Versão 1.1, veja apêndice A. O documento visão é construído durante a disciplina de Requisitos. Se houver necessidade o leitor pode recordar os tópicos 4.1.1 e 4.2.2 do Capítulo 4.
- Glossário, veja apêndice B. O Glossário é bastante utilizado em projetos maiores, neste trabalho o uso do glossário foi mínimo.
- Lista de Riscos, veja apêndice C. Os riscos sempre devem ser documentados e conforme é minimizado, a lista de riscos deve ser atualizada.
- Caso de Negócio, veja apêndice D.
- Plano de Desenvolvimento de Software, veja apêndice E. Este artefato é atualizado e incrementado durante todo o projeto, inclusive na fase de transição.
- Plano de Iteração, veja apêndice F. Neste projeto todas as iterações foram planejadas.
- Especificação de Casos de Uso, veja apêndice G.
- Documento Visão Versão 1.2, veja apêndice H. Após a Especificação de Casos de Uso, foram descobertos novos requisitos, com isso foi gerado uma nova versão do artefato visão.

A Figura 6.2 retrata de forma geral como foi percorrido a primeira fase do projeto. Percebe-se que, a discplina de Modelagem de Negócios não foi utilizada, isso depende do projeto.

Para garantir a organização dos artefatos, foi criado uma estrutura de diretórios, onde cada diretório fase é composta pelas disciplinas do RUP. Veja a Figura 6.3.

As atualizações das versões dos artefatos foram feitas de acordo com as iterações, ou seja, se na fase de elaboração o documento visão é necessário ser atualizado, então é criado uma nova versão deste documento dentro do diretório disciplinas da fase de elaboração. Assim o processo de repete sempre que for necessário.

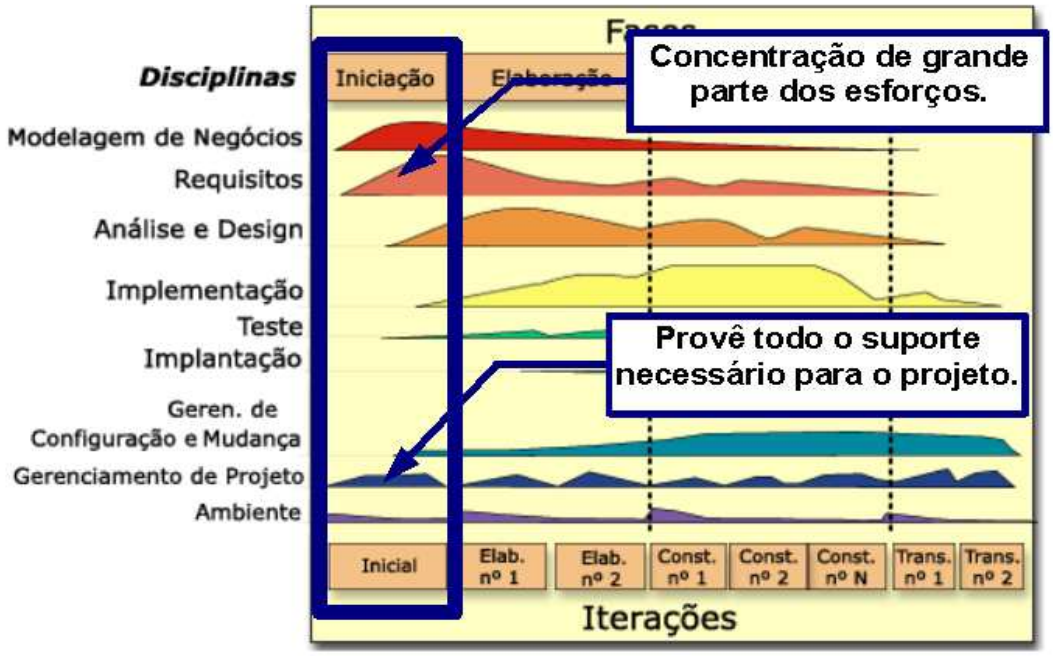

Figura 6.2: Visão geral da fase de Concepção.

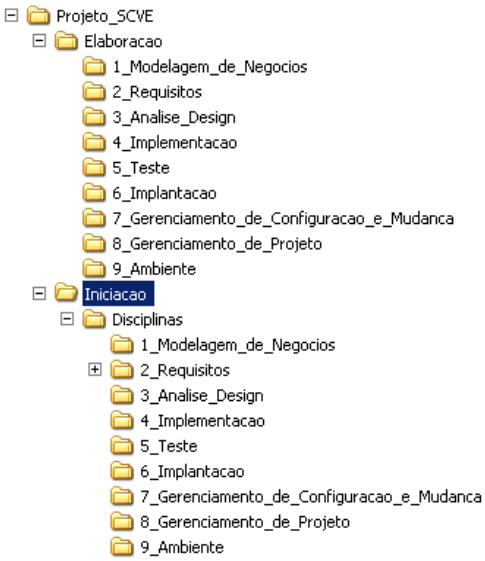

Figura 6.3: Organização dos Artefatos.

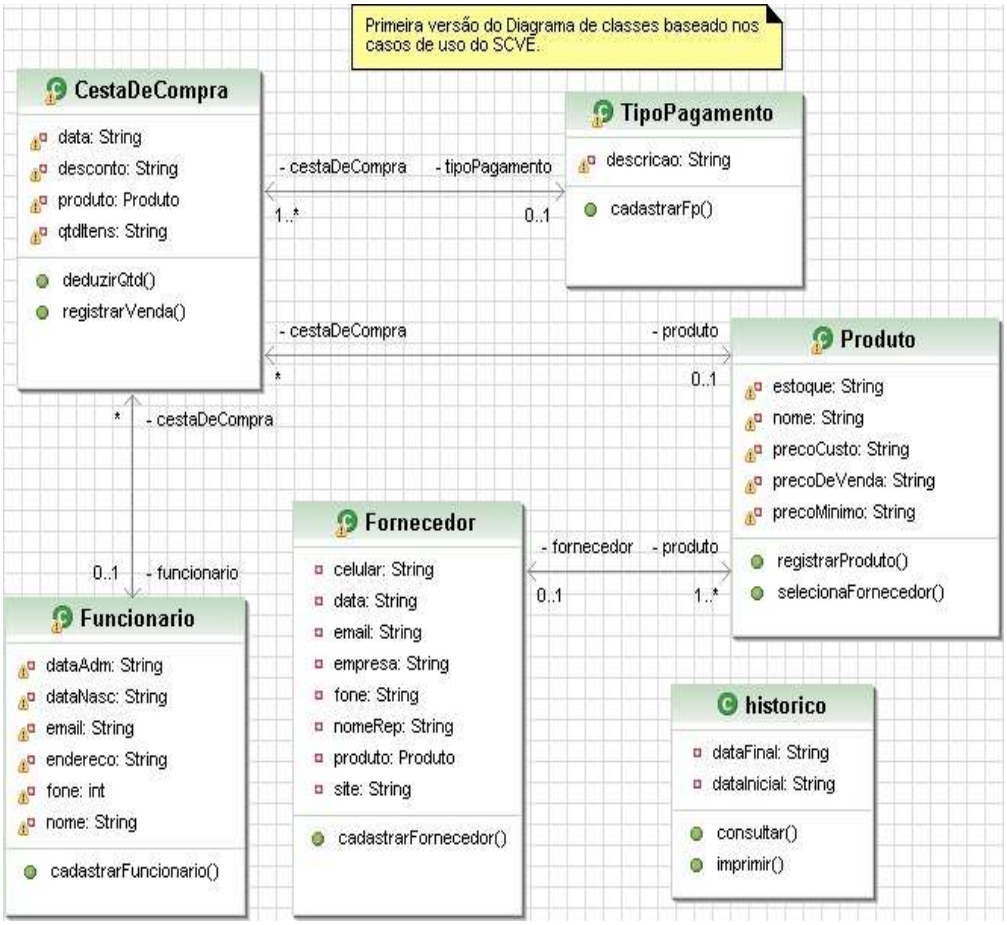

Figura 6.4: Diagrama de Classes Inicial.

6.2 Fase de Elaboração do Projeto SCVE

A primeira atividade realizada nesta fase foi a elaboração dos artefatos plano de iteração e ambiente de desenvolvimento, onde foram definidas as ferramentas necessárias para o desenvolvimento do projeto, veja os apêndices I e J.

Grande parte dos esforços foram dedicados na disciplina de Análise e *Design*. Com base nas especificações de casos de uso, foi eleborado o primeiro diagrama de classes conforme visto na Figura 6.4.

Em seguida foi desenvolvido o diagrama de sequência com o objetivo de melhorar o diagrama de classes. Figura 6.5.

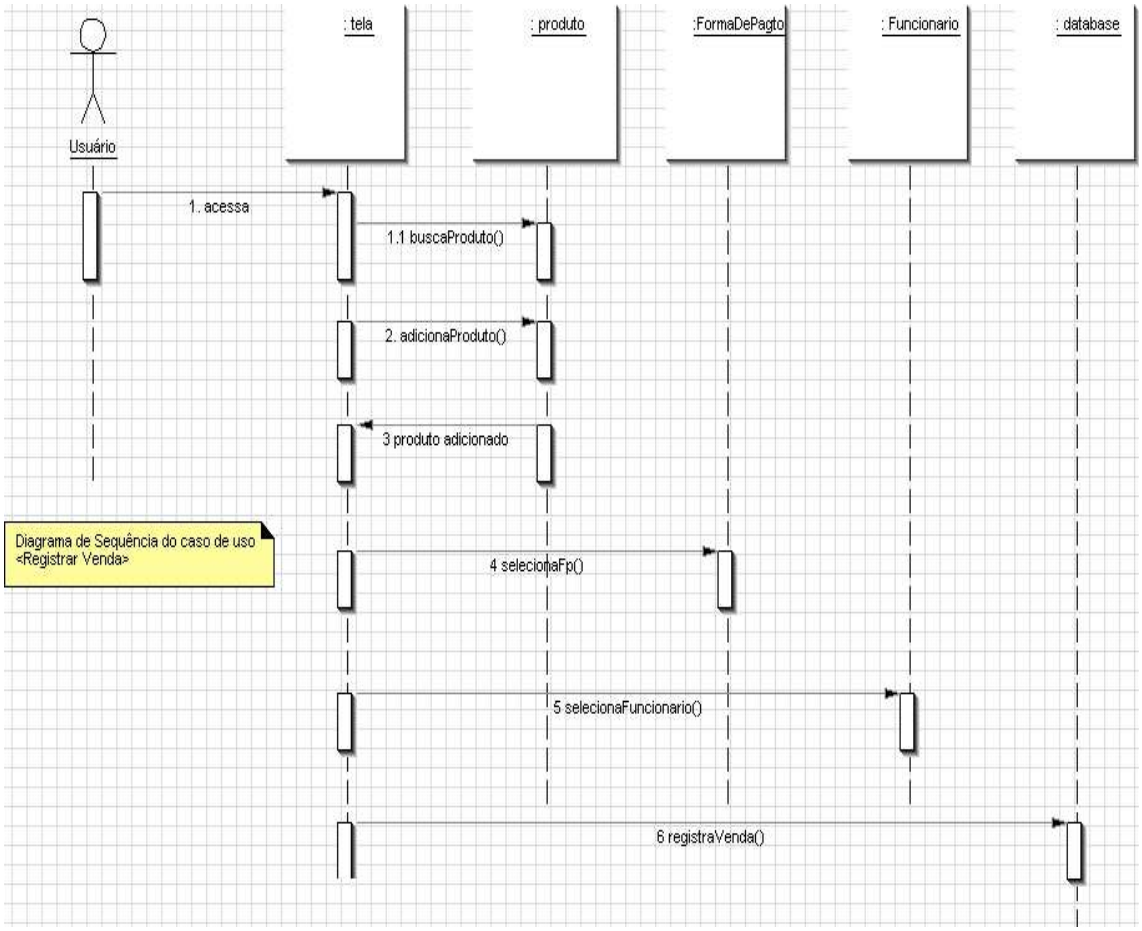

Figura 6.5: Diagrama de Sequência.

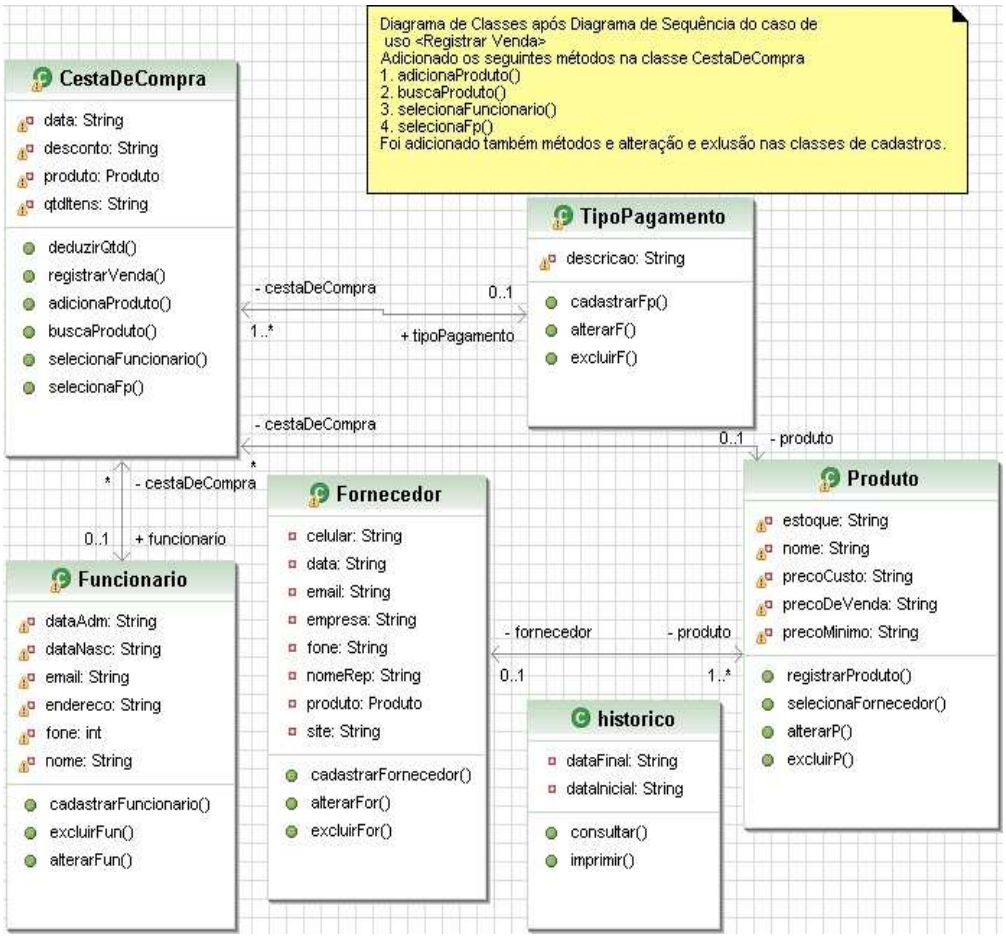

Figura 6.6: Diagrama de Classes Final.

Após o diagrama de sequência, foi definido uma nova versão do diagrama de classes. Figura 6.6.

Com a finalização do diagrama de classes, o próximo passo foi o desenvolvimento do modelo de dados conforme mostra a Figura 6.7.

Uma vez definido o ambiente e de posse do diagrama de classes final e modelo de dados, partiu-se para a elaboração do artefato arquitetura, veja apêndice K. O passo seguinte foi atualizar o plano de desenvolvimento de *software*, veja apêndice L.

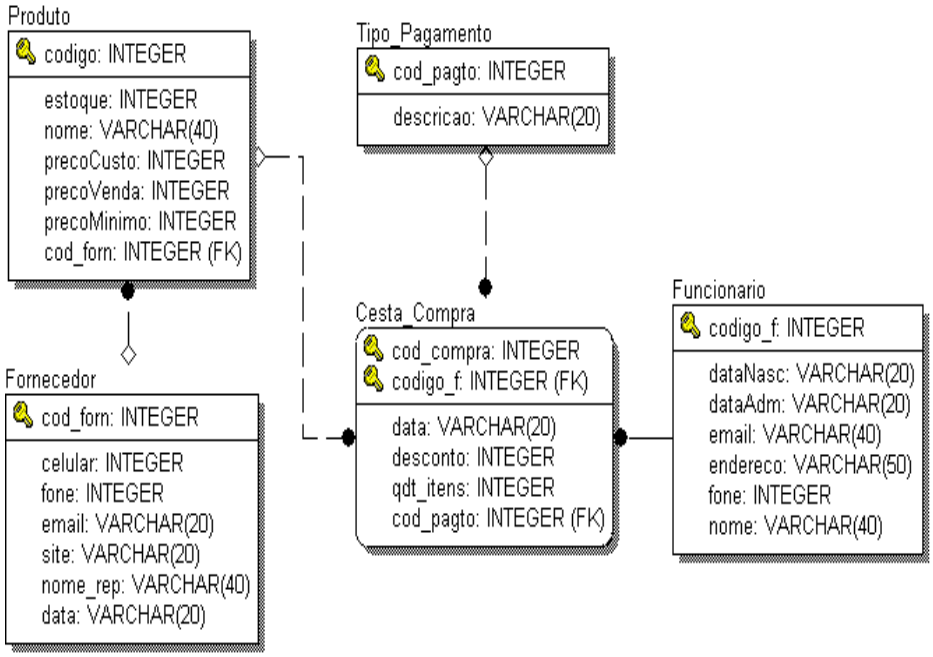

Figura 6.7: Modelo de dados do projeto.

Capítulo 7

Conclusão

A meta principal no desenvolvimento de um *software* é simplesmente fornecer o que o cliente deseja. Sendo assim, um dos pontos fundamentais para um correto processo de desenvolvimento de *software*, é ter conhecimento o suficiente para aplicar os conceitos e processos da metodologia adotada. O RUP por ser um processo muito extenso e abrangente, demonstra uma impressão de ser completamente inadequado para pequenos projetos, o que na realidade não é verdade, pois através deste trabalho demonstrou-se a utilização do RUP em um pequeno projeto de *software*.

Uma das grandes dificuldades encontradas no início do desenvolvimento do projeto foi definir o que realmente era necessário para o projeto em questão, principalmente em termos de artefatos. O RUP propõe alguns artefatos muito extensos, mas ao mesmo tempo é possível aproveitar apenas alguns ítens de um determinado artefato que se enquadre com o projeto proposto. Tudo depende da abrangência e complexidade do projeto. Outro fator importante foi em relação às disciplinas do RUP, onde descobriu-se que, nem todos os projetos passam por todas as disciplinas. Foi o que aconteceu neste projeto, a disciplina de Modelagem de Negócios não foi utilizada. Outra disciplina que talvez não pudesse ser utilizada é a de Gerenciamento de Configuração e Mudança, caso não houvesse mudanças durante a fase de construção. Desta forma foi utilizado apenas o suficiente do RUP para suprir as necessidades do projeto.

Com a prática, a equipe amadureceu muito em relação a este trabalho, pois conforme as

atividades vão sendo desenvolvidas, ganhou-se experiência no processo de desenvolvimento e ao mesmo tempo fazendo uso das melhores práticas de desenvolvimento adotadas pelo RUP, o que permite reduzir em horas uma determinada atividade que na primeira vez poderia levar dois dias. É como andar de carro pela primeira vez, nunca se consegue fazer uma conversão correta e principalmente entender perfeitamente as regras de trânsito e preferência. No RUP quanto mais se estuda mais se aprende, e a prática é fundamental para o entendimento.

Diante de todo o estudo realizado, o RUP foi adaptado de acordo com o projeto, ou seja, uma empresa de desenvolvimento pode configurar o RUP ou criar sua própria versão do Processo Unificado de acordo com o patamar de seus projetos, um desenvolvedor também pode customizar o RUP de acordo com o seu mercado (por exemplo: aquele que desenvolve apenas sistemas para o comércio em geral) em relação aos *softwares* desenvolvidos.

Enfim, atualmente o RUP é sem dúvida nenhuma um dos melhores processos a ser adotado como metodologia de desenvolvimento de *software*, independente do tamanho do projeto, bastando apenas ter um *know-how* o suficiente para poder aplicá-lo.

Apêndice A

Sistema de Controle de Vendas e Estoque	Versão: 1.1
D:/ProjetoSCVE/Iniciacao/Disciplinas/2.Requisitos	Data: 01/10/2006

Documento Visão – Versão 1.1

Histórico da Revisão

01/10/2006	1.0	Criação do documento.	Lidimon Cristiano
04/10/2006	1.1	Detalhamento dos recursos do produto. Item 5.	Lidimon Cristiano

1. Introdução

A finalidade deste documento é coletar, analisar e definir as características gerais do Sistema de Controle de Vendas e Estoque (SCVE) de acordo com as necessidades do cliente. Ele enfoca os recursos de que os envolvidos e usuários-alvo precisam e mostra por que essas necessidades existem.

1.1 Referências

Anexo 1.

2. Posicionamento

2.1 Descrição do Problema

O problema	de registrar as vendas em uma planilha eletrônica, onde é necessário sempre estar digitando o nome do produto, a quantidade, a inicial do vendedor, o valor e a forma de pagamento.
afeta	a qualidade do negócio em geral.
cujo impacto é	a impossibilidade de obter informações sobre o histórico de vendas realizadas tanto por produto quanto por período.
uma boa solução seria	a utilização de um sistema que constrolasse as vendas e o estoque de forma automática e possibilitasse a obtenção de relatórios de vendas por período e produto.

2.2 Sentença de Posição do Produto

Para	a Locadora de Games.
Que	precisa automatizar os processos e vendas e controlar o estoque.
O SCVE	é um software (programa de computador).
Ao contrário de	software complexos com uma infinidade de recursos não necessários para o negócio atual.
Nosso produto	será de fácil utilização e diminuirá o tempo gasto no registro das vendas dando baixa automaticamente no estoque e possibilitará a visualização de relatórios com informações diversas sobre o histórico de vendas.

3. Descrições dos Envolvidos e Usuários

3.1 Resumo dos Envolvidos

Nome	Descrição	Responsabilidades
Gerente	Responsável por gerenciar o negócio e usar o sistema.	Informar as necessidades do sistema para o(s) analista(s).

3.2 Resumo dos Usuários

Nome	Descrição	Responsabilidades	Envolvido
Funcionário	Funcionário que irá usar o sistema.	Realizar vendas, consultar relatório de vendas.	Representado pelo gerente.

3.3 Ambiente do Usuário

Atualmente, a empresa utiliza um computador com o sistema operacional MS Windows 2000. O computador é utilizado pelo gerente e dois funcionários. A planilha que está sendo utilizada para registrar as vendas não será mais utilizada, pois não atende as necessidades atuais do negócio. A empresa irá precisar de uma impressora, podendo ser Lazer, Matricial ou Jato de Tinta caso queira fazer a impressão de relatórios.

3.4 Resumo das Principais Necessidades dos Envolvidos ou Usuários

Necessidade	**Prioridade**	**Preocupações**	**Solução Atual**	**Soluções Propostas**
Registrar as vendas	Alta	O sistema deve possibilitar o registro da venda de um produto de forma rápida.	As vendas são registradas em uma planilha eletrônica.	Permitir armazenar as informações relativas ao registro de vendas em um sistema de informações.
Controlar estoque na medida em que os produtos são vendidos	Alta	O sistema deve subtrair a quantidade do estoque no momento em que à venda do produto for feita.	Nenhuma	Permitir o controle do estoque de forma automatizada.
Cadastrar produto	Alta	Nenhuma	Nenhuma	Permitir cadastrar o produto com suas informações relevantes.
Consultar estoque	Média	Nenhuma	Nenhuma	Permitir a geração de relatório de produtos em estoque.
Consultar / Imprimir histórico de vendas e produtos de forma detalhada	Média	Nenhuma	Nenhuma	Permitir a geração de relatório de vendas.

3.5 Alternativas e Concorrência

Utilização de outro sistema que ofereça a solução para os problemas listados ou continuar os trabalhos sem a utilização de um sistema de informações.

4. Visão Geral do Produto

4.1 Perspectiva do Produto

O produto será um sistema independente, que não dependerá de outros programas. O usuário irá utilizá-lo através de interfaces (telas) para gerenciar o sistema e manter as informações armazenadas.

4.2 Suposições e Dependências

O sistema será feito para ser executado com sistema operacional MS Windows ou qualquer outro que suporte a tecnologia Java. Como banco de dados para o sistema, será utilizado uma versão open source (gratuita) para não ser necessário que o cliente tenha custos na compra um sistema de gerenciamento de banco de dados proprietário.

5. Recursos do Produto

O sistema deverá permitir o registro de vendas através de uma tela, onde o usuário deverá selecionar o produto, o vendedor, a forma de pagamento e a quantidade a ser vendida.

O sistema deve controlar o estoque no momento em que o usuário registrar uma venda.

O sistema deve permitir o cadastro de produtos através de uma tela, onde o usuário terá que entrar com os dados do produto e depois cadastrar.

O sistema deve permitir o cadastro de fornecedores através de uma tela, onde o usuário terá que entrar com os dados do fornecedor e depois cadastrar.

O sistema deve permitir obter o histórico de vendas realizadas por período ou produto através de uma tela, onde o usuário poderá manipular a pesquisa através da digitação de datas, ou poderá buscar por um produto qualquer para saber o histórico de vendas do produto solicitado.

O sistema deve permitir obter o histórico de produtos em estoque através de uma tela, onde o usuário poderá visualizar todos os produtos no estoque.

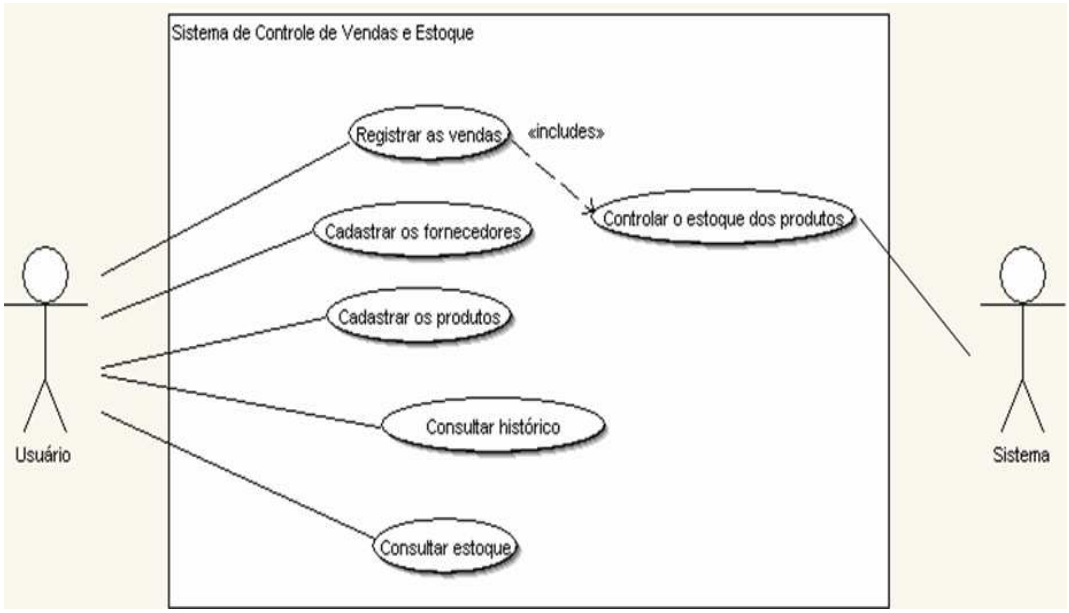

Figura 7.1: Casos de Uso Versão 1.1.

6. Outros Requisitos do Produto

Será entregue apenas o executável do sistema. Não está previsto a criação de manuais de instalação ou utilização. O produto poderá ser utilizado em qualquer loja da empresa, sem a necessidade de pagamentos adicionais por isso. Não haverá gerenciamento de usuários, ou seja, qualquer pessoa utilizando o computador poderá utilizar todos os recursos disponíveis no sistema.

7. Visão geral do Sistema - Modelo Conceitual

8. Anexos

O anexo abaixo tem por finalidade coletar informações pertinentes de como são realizados os processos atualmente.

Anexo 1. O registro de vendas da Locadora de Games é feito atualmente em uma planilha eletrônica conforme a imagem abaixo. Na planilha é especificada a data em que determinado produto foi vendido, o histórico (descrição), forma de pagamento, inicial do nome do vendedor e valor do produto multiplicado por sua quantidade caso seja mais de uma unidade do mesmo produto. Esta forma de registrar as vendas impossibilita as necessidades de obter um histórico

Figura 7.2: Exemplo atual de registro de vendas.

tanto por vendas realizadas em determinado período quanto por produto segundo o gerente responsável.

Apêndice B

Sistema de Controle de Vendas e Estoque	Versão: 1.1
D:/ProjetoSCVE/Iniciacao/Disciplinas/2.Requisitos	Data: 01/10/2006

Glossário

Histórico da Revisão

Data	Versão	Descrição	Autor
01/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

1. Introdução

1.1 Finalidade

O glossário contém as definições de funcionalidade do SCVE. Este glossário será expandido durante toda a vida do projeto.

1.2 Escopo

Este glossário trata de todos os termos que possuem significados específicos neste projeto.

2 Definições

2.1 Registrar venda

Registrar venda é quando um cliente paga pelo produto que comprou. Logo em seguida o funcionário registra a venda.

Apêndice C

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Iniciacao/Disciplinas/8.Gerenciamento de Projeto	Data: 03/10/2006

Lista de Riscos

Histórico da Revisão

Data	Versão	Descrição	Autor
03/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

1. Introdução

1.1 Finalidade

Este documento descreve os riscos conhecidos do projeto.

1.2 Escopo

Identificar riscos e as características críticas do produto.

1.3 Definições, Acrônimos e Abreviações.

Consulte documento Glossário.

1.4 Referências

Documento Visão

1.5 Visão Geral

Os riscos conhecidos na data de publicação deste documento são listados abaixo, junto com as estratégias de diminuição de cada risco.

2. Riscos

2.1 Risco de Programação

Classificação ou Gravidade do Risco	Alto
Descrição	Como este é o primeiro projeto da equipe, a mesma nunca desenvolveu algum aplicativo em Java, somente em Delphi.
Impactos	A fase de construção fica comprometida.
Estratégia de Diminuição	As iterações devem ser calculadas cuidadosamente para disponibilizar mais tempo durante a fase de construção.
Plano de Contingência	Treinamento em ambiente java.

Apêndice D

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Iniciacao/Disciplinas/8.Gerenciamento de Projeto	Data: 05/10/2006

Caso de Negócio

Histórico da Revisão

Data	Versão	Descrição	Autor
05/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

1. Introdução

1.1 Finalidade

Fornecer as informações necessárias do ponto de vista do negócio para determinar se é viável investir no projeto ou não.

1.2 Escopo

Este documento esta associado ao projeto SCVE.

1.3 Definições, Acrônimos e Abreviações

Consulte Documento Glossário para maiores informações.

1.4 Referências

Documento Visão.

2 Visão Geral

As próximas seções contêm informações sobre o produto, contexto do negócio, objetivos do produto, estimativas financeiras e restrições do projeto SCVE.

2.1 Descrição do Produto

O Sistema de Controle de Vendas e Estoque (SCVE) é um sistema que atende a comércios que trabalham com vendas de produtos. O SCVE permite registrar todas as vendas realizadas e com base nestas informações, o SCVE gera diversos relatórios para consultas. O sistema também controla a quantidade de produtos no estoque à medida que cada produto é vendido, além disso permite o cadastro dos produtos, funcionários e fornecedores. Para maiores informações sobre a funcionalidade do sistema consulte o Documento Visão.

2.2 Contexto do Negócio

O SCVE será desenvolvido para substituir o sistema de planilhas eletrônicas da Locadora de Games, por não atender as necessidades do negócio. É um sistema que será desenvolvido especificamente para atender as necessidades do negócio em si, podendo ser adaptado para qualquer empresa que possui um negócio similar.

3. Objetivos do Produto

Possibilitar que a empresa Locadora de Games possa melhor gerenciar o seu negócio de vendas.

4. Estimativas Financeiras

Para a equipe de desenvolvimento. O objetivo do desenvolvimento deste projeto é apenas para atender as necessidades da Locadora de Games, ou seja, a equipe esta desenvolvendo o projeto sem cobrar nenhum valor. Após a conclusão será feito um estudo de viabilidade para a comercialização do produto com o objetivo de recuperar os esforços realizados neste projeto. Com o SCVE a Locadora de Games poderá atender a demanda de seus clientes através das informações que o sistema irá gerar conforme as vendas vão sendo realizadas e armazenadas no sistema. Desta forma a empresa terá informações o suficiente para prever qual o período que determinado produto(s) é mais vendido e poderá comprar uma quantidade suficiente de determinado item para atender a demanda pelo produto em questão. Com isso a empresa

obterá maior número de vendas, poderá traçar estratégias de marketing e aumentará a sua receita.

6. Restrições

Esse projeto possui um grande risco pelo fato de que este é o primeiro projeto da equipe, sendo viável o desenvolvimento do projeto para a aquisição de experiência.

Apêndice E

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Iniciacao/Disciplinas/8.Gerenciamento de Projeto	Data: 02/10/2006

Plano de Desenvolvimento de Software

Histórico da Revisão

Data	Versão	Descrição	Autor
02/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

1. Introdução

1.1 Finalidade

A finalidade deste Plano de Desenvolvimento de Software é definir as atividades de desenvolvimento no que diz respeito às fases e iterações necessárias para a implementação do SCVE para a Locadora de Games.

1.2 Escopo

Este Plano de Desenvolvimento de Software descreve o plano geral a ser usado pela equipe para desenvolver o sistema. Os detalhes de iterações individuais serão descritos nos Planos de Iteração.

1.3 Definições, Acrônimos e Abreviações

Consulte documento Glossário.

1.4 Referências

Documento Visão, Lista de Riscos, Planos de Iteração, Caso de Negócio.

2. Visão Geral do Projeto

2.1 Finalidade, escopo e objetivos do projeto

Este projeto implementará um Sistema de Controle de Vendas e Estoque para substituir o antigo controle de vendas, que é realizado através de planilhas eletrônicas. Consulte Documento Visão para maiores detalhes.

2.2. Suposições e Restrições

Nenhuma

2.3. Produtos Liberados do Projeto

Os produtos liberados a seguir serão produzidos durante o projeto:

Documento Visão

Plano de Desenvolvimento de Software

Caso de Negócio

Glossário

Especificação de Casos de Uso

Plano de Iteração

Lista de Riscos

2.4 Evolução do Plano de Desenvolvimento de Software

O Plano de Desenvolvimento de Software sempre será revisado antes do início de cada fase de iteração. As datas previstas para o final de cada fase esta prevista a seguir.

Fase	**Término**
Concepção	06/10/06
Elaboração	Será definida após o término da iniciação.

3 Organização do Projeto

3.1 Estrutura Organizacional

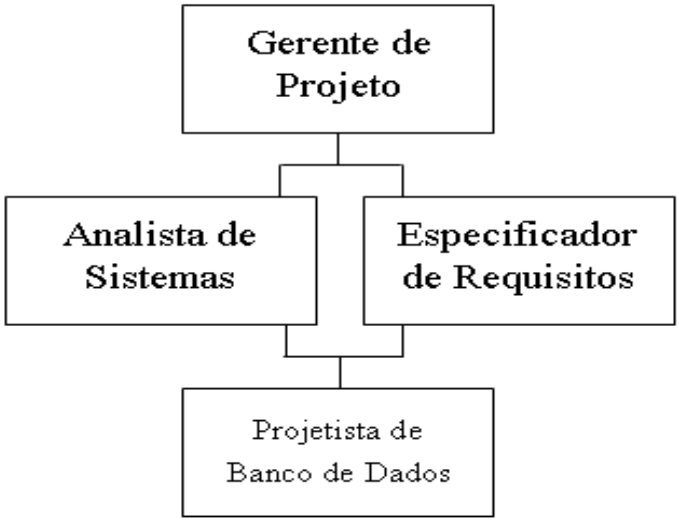

Figura 7.3: Estrutura Organizacional Versão 1.0.

A equipe de projeto da fase de iniciação e elaboração será organizada da seguinte maneira:

3.2 Interfaces Externas

O Gerente da Locadora de Games e os funcionários mais experientes irão informar os requisitos do sistema. O gerente irá testar as funcionalidades do sistema.

3.3 Papéis e Responsabilidades

A tabela a seguir mostra as funções representadas no diagrama anterior e suas respectivas responsabilidades.

Papel	**Responsabilidade**
Gerente de Projeto	O Gerente de Projeto aloca recursos, especifica prioridades, coordena as interações com os clientes e usuários e, geralmente, tenta manter a equipe de projeto centrada na meta correta. O Gerente de Projeto também estabelece um conjunto de práticas que garante a integridade e qualidade dos artefatos do projeto.
Analista de Sistemas	O papel analista de sistemas lidera é coordenar a identificação de requisitos e a modelagem de casos de uso, delimitando o sistema e definindo sua funcionalidade; por exemplo, estabelecendo quais são os atores e casos de uso existentes e como eles interagem.
Especificador de Requisitos	O papel especificador de requisitos detalha a especificação de uma parte da funcionalidade do sistema, descrevendo o aspecto Requisitos de um ou de vários casos de uso e outros requisitos de software de apoio.
Projetista (Design) de Banco de Dados	O papel designer de banco de dados é definir tabelas, índices, visões, restrições, triggers, procedimentos armazenados, parâmetros de armazenamento e outras construções específicas de um banco de dados necessárias para armazenar, recuperar e excluir objetos persistentes.

4. PROCESSO DE GERENCIAMENTO

4.1 Estimativas do Projeto

A fase de iniciação deste projeto deve durar uma semana.

4.2 Plano do Projeto

A fase de iniciação do projeto pode ser visualizada da seguinte maneira:

Fase	**Início**	**Fim**
Concepção	01/10/06	06-10-06

4.2.1 Plano de Fase

O desenvolvimento será conduzido usando uma abordagem com fases onde várias iterações ocorrem dentro de uma fase. As fases e a linha de tempo relativa são mostradas na tabela a seguir:

Fase	**Nº de Iterações**	**Início**	**Fim**
Fase de Iniciação	1	01/10/06	06/10/06
Fase de Elaboração	1	09/10/06	25/10/06

Os marcos que indicam o fim de cada fase podem ser vistos na tabela a seguir.

Descrição	**Marco**
Fase de Cencepção	A fase de concepção desenvolverá os requisitos do produto e estabelecerá o caso de negócio. Os principais casos de uso serão desenvolvidos, bem como este Plano de Desenvolvimento de Software.
Fase de Elaboração	A Fase de Elaboração analisará os requisitos e desenvolverá o protótipo de arquitetura. Após concluir a Fase de Elaboração, o projeto já estará concluído e pronto para a fase de construção.

4.2.2 Objetivos da Iteração

Fase	**Nº de Iterações**	**Descrição**	**Marcos Associados**
Concepção	Iteração Preliminar	Definir os principais requisitos do sistema, caso de negócio, iteração e o plano de desenvolvimento de software.	Aprovação do Caso de Negócio Identificação dos Riscos

4.2.3 Releases

Neste ponto nenhum release será apresentado.

4.2.4 Orçamento

Este projeto não possui orçamento, a experiência adquirida pela equipe servirá como forma de pagamento pelos esforços realizados.

Apêndice F

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Iniciacao/Disciplinas/8.Gerenciamento de Projeto	Data: 03/10/2006

Plano de Iteração

Histórico da Revisão

Data	Versão	Descrição	Autor
03/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

1. Introdução

1.1 Finalidade

Este Plano de Iteração descreve os planos detalhados para a Iteração Preliminar do Projeto. Durante essa iteração, serão definidos prazos, alocação de papeis, requisitos do sistema e criação de artefatos.

1.2 Escopo

Identificar riscos.

Identificar as características críticas do produto.

1.3 Definições, Acrônimos e Abreviações.

Consulte documento Glossário.

1.4 Referências

Documento Visão

2. Plano

A Iteração Preliminar desenvolverá os requisitos do produto e estabelecerá o caso de negócio para o SCVE. Os principais casos de uso serão desenvolvidos.

2.1 Tarefas da Iteração

A tabela a seguir ilustra as tarefas a serem realizadas com as respectivas datas de início e término. Como este é o primeiro projeto da equipe, cada tarefa será programada após o término da anterior.

Tarefa	**Disciplina**	**Início**	**Término**	**Papel responsável**
Início da Fase de Concepção	01/10/06	06/10/06		
Desenvolver Documento Visão	Requisitos	01/10/06	02/10/06	Lidimon
Desenvolver e manter Documento Glossário	Requisitos	01/10/06	Em expansão	Newton
Solicitar aprovação do Cliente para Documento Visão	Requisitos	02/10/06	02/10/06	Lidimon
Realizar Caso de Negócio	Gerenciamento de Projeto	03/10/06	03/10/06	Lidimon - Newton
Criação Plano de Iteração, Lista de Riscos	Gerenciamento de Projeto	04/10/06	04/10/06	Lidimon - Newton
Desenvolver Plano de Desenvolvimento de Software	Gerenciamento de Projeto	05/10/06	05/10/06	Lidimon - Newton
Especificação de Casos de Uso	Requisitos	06/10/06	06/10/06	Lidimon - Newton
Solicitar aprovação do Cliente - Atualizar Documento Visão	Requisitos	06/10/06	06/10/06	Lidimon
Atualizar Artefatos		06/10/06	06/10/06	Lidimon - Newton
Fim da Concepção		06/10/06	06/10/06	

Os seguintes artefatos serão gerados e revisados durante a Iteração desta fase.

Disciplina	**Artefato liberado durante a disciplina**	**Proprietário responsável**
Requisitos	Documento Visão	Lidimon
Requisitos	Especificação de Casos de Uso	Lidimon - Newton
Requisitos	Glossário	Newton
Gerenciamento de Projeto	Plano de Iteração	Lidimon - Newton
Gerenciamento de Projeto	Plano de Desenvolvimento de Software	Lidimon - Newton
Gerenciamento de Projeto	Lista de Riscos do Projeto	Newton
Gerenciamento de Projeto	Caso de Negócio	Lidimon - Newton

3 Casos de Uso

Durante a Iteração Preliminar, todos os casos de uso e atores importantes serão identificados. Os cenários básicos e os cenários alternativos de cada caso de uso serão determinados e documentados nas Especificações de Casos de Uso.

4 Critérios de Avaliação

A meta principal da Iteração Preliminar é definir o sistema no nível de detalhamento necessário para permitir uma avaliação econômica mais completa com relação à viabilidade do projeto a partir de uma perspectiva de negócio. Com a finalização da iteração, uma revisão do Caso de Negócio decidirá sobre a Viabilidade / Inviabilidade do projeto.

Apêndice G

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Iniciacao/Disciplinas/2.Requisitos	Data: 03/10/2006

Especificação de Caso de Uso

Histórico da Revisão

Data	Versão	Descrição	Autor
03/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

Especificação de Caso de Uso Registrar Venda

1. Nome do Caso de Uso

Registrar Venda (UC001)

1.1 Breve Descrição

Este caso de uso ocorre quando o cliente compra algum produto, neste momento o funcionário registra a venda do produto no sistema.

2. Cenário Básico

1. Usuário através de uma tela busca o(s) produto(s) que o cliente irá comprar.

2. Usuário adiciona o produto que o cliente esta comprando.

3. O funcionário deve informar no sistema:

3.1 Aquantidade de itens para cada produto.

3.2 Selecionar a forma de pagamento.

3.3 Selecionar o nome do funcionário que esta realizando a operação.

3.4 Informar o valor de desconto se houver.

4. O funcionário registra a venda.

5. O sistema deduz a quantidade de produto no estoque (ver Caso de Uso Controlar Estoque) e registra a data da venda.

2.1 Cenário Alternativo

2. O sistema verifica a quantidade em estoque, compara com a quantidade informada e emite um aviso para o funcionário caso a quantidade no estoque seja menor que a quantidade solicitada.

2.1 Se o funcionário não selecionar a forma de pagamento ou funcionário responsável pela operação, o sistema emite um aviso.

3. Pré-Condições

É necessário que tenha pelo menos um produto cadastrado no sistema e que a quantidade deste produto em estoque seja igual ou maior que 1.

4. Atores envolvidos

Usuário (Funcionário ou Gerente)

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Prever Caso de Uso Cadastrar Funcionário, Cadastrar Produto, Cadastrar Forma de Pagamento.

Especificação de Caso de Uso Cadastrar Funcionário

1. Nome do Caso de Uso

Cadastrar Funcionário (UC002)

1.1 Breve Descrição

Este caso de uso ocorre uma única vez quando é feito o cadastramento dos funcionários que irão utilizar o sistema.

2. Cenário Básico

1. Usuário informa os seguintes dados: Nome, Data de Nascimento, Data de Admissão, Telefone, Endereço, E-mail.

2. Sistema registra funcionário.

2.1 Cenário Alternativo

2. Caso os dados Nome, Data de Nascimento e Data de Admissão estiverem nulos, sistema emite mensagem e não registra o funcionário.

3. Pré-Condições

Nenhuma.

4. Atores envolvidos

Usuário

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Nenhuma

Especificação de Caso de Uso Cadastrar Forma de Pagamento

1. Nome do Caso de Uso

Cadastrar Forma de Pagamento (UC003)

1.1 Breve Descrição

Este caso de uso ocorre sempre após a instalação do sistema ou no decorrer do tempo caso seja necessário adicionar novas formas de pagamento.

2. Cenário Básico

1. Usuário informa o tipo de pagamento (por exemplo: à vista, cheque etc) para que seja possível selecionar o mesmo quando for registrar a venda de algum produto.

2. Usuário registra forma de pagamento.

2.1 Cenário Alternativo

2. Caso o tipo de pagamento esteja em branco, o sistema emite mensagem.

3. Pré-Condições

Nenhuma.

4. Atores envolvidos

Usuário

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Nenhuma

Especificação de Caso de Uso Cadastrar Produto

1. Nome do Caso de Uso

Cadastrar Produto (UC004)

1.1 Breve Descrição

Este caso de uso ocorre sempre quando a Locadora de Games adquire de seu fornecedor novos produtos para revender.

2. Cenário Básico

1. Usuário informa os seguintes dados:

1.1 Descrição do produto (nome)

1.2 Preço de custo unitário.

1.3 Preço de venda.

1.4 Preço mínimo de venda.

1.5 Seleciona o fornecedor do produto.

1.6 Informa estoque (quantidade disponível).

2. Usuário registra produto.

2.1 Cenário Alternativo

2. Caso alguns dos dados Nome do Produto, Preço de venda e estoque estejam nulos, o sistema emite aviso.

3. Pré-Condições

É necessário que o fornecedor esteja cadastrado (Caso de Uso Cadastrar Fornecedor) para relacionar com o produto em questão, porém na hora de cadastrar algum produto não é obrigatório selecionar fornecedor.

4. Atores envolvidos

Usuário

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Nenhuma

Especificação de Caso de Uso Cadastrar Fornecedor

1. Nome do Caso de Uso

Cadastrar Produto (UC005)

1.1 Breve Descrição

Este caso de uso ocorre quando o usuário da Locadora de Games deseja cadastrar os fornecedores de seus produtos.

2. Cenário Básico

1. Usuário informa os seguintes dados: Empresa, Nome Representante, Telefone, Celular, E-mail, Site, Data.

2. Usuário registra fornecedor.

2.1 Cenário Alternativo

2. Casos os dados Empresa ou Nome Representante estejam nulos, sistema emite mensagem.

3. Pré-Condições

Nenhuma

4. Atores envolvidos

Usuário

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Nenhuma

Especificação de Caso de Uso Controlar Estoque

1. Nome do Caso de Uso

Controlar Estoque (UC006)

1.1 Breve Descrição

Este caso de uso ocorre sempre quando a venda de algum produto é realizada, ou seja, na medida em que algum produto é vendido, sua quantidade é subtraída no estoque.

2. Cenário Básico

1. Usuário realiza caso de uso Registrar Vendas.

2. Sistema deduz do estoque a quantidade vendida.

2.1 Cenário Alternativo

3. Pré-Condições

Nenhuma

4. Atores envolvidos

Usuário

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Nenhuma

Especificação de Caso de Uso Consultar Histórico

1. Nome do Caso de Uso

Controlar Estoque (UC007)

1.1 Breve Descrição

Este caso de uso ocorre quando o usuário deseja consultar o histórico de algum produto ou estoque.

2. Cenário Básico

1. Usuário através de uma tela seleciona as seguintes opções para obter o histórico:

1.1 Lista de produto em estoque (mostra o nome do produto e a quantidade em estoque)

1.2 Lista de produtos vendidos. (mostra o nome do produto, quantidade vendida, data e preço de venda)

2. Usuário entra com uma data inicial e data final para pesquisa.

3. Resultado é exibido na tela.

4. Usuário solicita impressão.

5. Sistema imprime relatório.

2.1 Cenário Alternativo

3. Pré-Condições

Nenhuma

4. Atores envolvidos

Usuário

5. Requisitos Especiais

Durante a próxima iteração poderá haver requisitos especiais.

6. Observação

Nenhuma

Apêndice H

Sistema de Controle de Vendas e Estoque	Versão: 1.2
D:/ProjetoSCVE/Iniciacao/Disciplinas/2.Requisitos	Data: 06/10/2006

Documento Visão – Versão 1.2

Histórico da Revisão

01/10/2006	1.0	Criação do documento.	Lidimon Cristiano
04/10/2006	1.1	Detalhamento dos recursos do produto. Item 5.	Lidimon Cristiano
06/10/2006	1.2	Adicionado os casos de uso Cadastrar Funcionário e Cadastrar Forma de Pagamento no item 3.4. Diagrama de casos de uso alterado, item 7. Alteração do Item 6.	Lidimon Cristiano

1. Introdução

A finalidade deste documento é coletar, analisar e definir as características gerais do Sistema de Controle de Vendas e Estoque (SCVE) de acordo com as necessidades do cliente. Ele enfoca os recursos de que os envolvidos e usuários-alvo precisam e mostra por que essas necessidades existem.

1.1 Referências

Anexo 1.

2. Posicionamento

2.1 Descrição do Problema

O problema	de registrar as vendas em uma planilha eletrônica onde é necessário sempre estar digitando o nome do produto, a quantidade, a inicial do vendedor, o valor e a forma de pagamento.
afeta	a qualidade do negócio em geral.
cujo impacto é	a impossibilidade de obter informações sobre o histórico de vendas realizadas tanto por produto quanto por período.
uma boa solução seria	a utilização de um sistema que constrolasse as vendas e o estoque de forma automática e possibilitasse a obtenção de relatórios de vendas por período e produto.

2.2 Sentença de Posição do Produto

Para	a Locadora de Games.
Que	precisa automatizar os processos e vendas e controlar o estoque.
O SCVE	é um software (programa de computador).
Ao contrário de	software complexos com uma infinidade de recursos não necessários para o negócio atual.
Nosso produto	será de fácil utilização e diminuirá o tempo gasto no registro das vendas dando baixa automaticamente no estoque e possibilitará a visualização de relatórios com informações diversas sobre o histórico de vendas.

3. Descrições dos Envolvidos e Usuários

3.1 Resumo dos Envolvidos

Nome	Descrição	Responsabilidades
Gerente	Responsável por gerenciar o negócio e usar o sistema.	Informar as necessidades do sistema para o(s) analista(s).

3.2 Resumo dos Usuários

Nome	Descrição	Responsabilidades	Envolvido
Funcionário	Funcionário que irá usar o sistema.	Realizar vendas, consultar relatório de vendas.	Representado pelo gerente.

3.3 Ambiente do Usuário

Atualmente, a empresa utiliza um computador com o sistema operacional MS Windows 2000. O computador é utilizado pelo gerente e dois funcionários. A planilha que está sendo utilizada para registrar as vendas não será mais utilizada, pois não atende as necessidades atuais do negócio. A empresa irá precisar de uma impressora, podendo ser Lazer, Matricial ou Jato de Tinta caso queira fazer a impressão de relatórios.

3.4 Resumo das Principais Necessidades dos Envolvidos ou Usuários

Necessidade	**Prioridade**	**Preocupações**	**Solução Atual**	**Soluções Propostas**
Registrar as vendas	Alta	O sistema deve possibilitar o registro da venda de um ou vários produtos de forma rápida.	As vendas são registradas em uma planilha eletrônica.	Permitir armazenar as informações relativas ao registro de vendas em um sistema de informações.
Controlar estoque na medida em que os produtos são vendidos	Alta	O sistema deve subtrair a quantidade do estoque no momento em que à venda do produto for feita.	Nenhuma	Permitir o controle do estoque de forma automatizada.
Cadastrar produto	Alta	Nenhuma	Nenhuma	Permitir cadastrar o produto com suas informações relevantes.
Consultar estoque	Média	Nenhuma	Nenhuma	Permitir a geração de relatório de produtos em estoque.
Consultar / Imprimir histórico de vendas e produtos de forma detalhada	Média	Nenhuma	Nenhuma	Permitir a geração de relatório de vendas.

Cadastrar funcionário	Média	Nenhuma	Digitação da inicial do nome do funcionário em cada venda realizada.	Permitir selecionar o nome do funcionário quando o mesmo for efetuar alguma venda.
Cadastrar forma de pagamento	Média	Nenhuma	É digitada a forma de pagamento sempre quando um produto é vendido.	Permitir selecionar a forma de pagamento.

3.5 Alternativas e Concorrência

Utilização de outro sistema que ofereça a solução para os problemas listados ou continuar os trabalhos sem a utilização de um sistema de informações.

4. Visão Geral do Produto

4.1 Perspectiva do Produto

O produto será um sistema independente, que não dependerá de outros programas. O usuário irá utilizá-lo através de interfaces (telas) para gerenciar o sistema e manter as informações armazenadas.

4.2 Suposições e Dependências

O sistema será feito para ser executado com sistema operacional MS Windows ou qualquer outro que suporte a tecnologia Java. Como banco de dados para o sistema, será utilizado uma versão open source (gratuita) para não ser necessário que o cliente tenha custos na compra um sistema de gerenciamento de banco de dados proprietário.

5. Recursos do Produto

O sistema deverá permitir o registro de vendas através de uma tela, onde o usuário deverá selecionar o produto, o vendedor, a forma de pagamento e a quantidade a ser vendida.

O sistema deve controlar o estoque no momento em que o usuário registrar uma venda.

O sistema deve permitir o cadastro de produtos através de uma tela, onde o usuário terá que entrar com os dados do produto e depois cadastrar.

O sistema deve permitir o cadastro de fornecedores através de uma tela, onde o usuário terá que entrar com os dados do fornecedor e depois cadastrar.

O sistema deve permitir obter o histórico de vendas realizadas por período ou produto através de uma tela, onde o usuário poderá manipular a pesquisa através da digitação de datas, ou poderá buscar por um produto qualquer para saber o histórico de vendas do produto solicitado.

O sistema deve permitir obter o histórico e impressão de produtos em estoque através de uma tela, onde o usuário poderá visualizar todos os produtos no estoque.

O sistema deve permitir o cadastro de funcionários através de uma tela, onde o usuário terá que entrar com os dados do produto e depois cadastrar.

6. Outros Requisitos do Produto

O sistema será compatível tanto com Windows quanto Linux. Em relação aos requisitos de Hardware, recomenda-se que o computador tenha no mínimo 128MB de Ram e processador 450 Mhz.

Será entregue apenas o executável do sistema. Não está previsto a criação de manuais de instalação ou ajuda de utilização, pois se trata de um sistema simples e de fácil aprendizado. Não haverá gerenciamento de usuários, ou seja, qualquer pessoa utilizando o computador poderá utilizar todos os recursos disponíveis no sistema.

7. Visão geral do Sistema - Modelo Conceitual

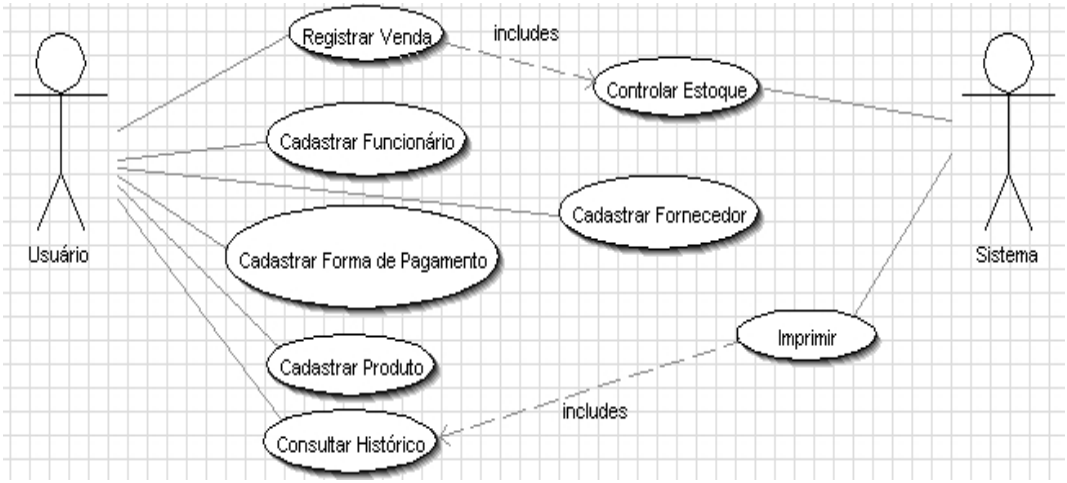

Figura 7.4: Casos de Uso Versão 1.2.

Apêndice I

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Elaboracao/Disciplinas/9.Ambiente	Data: 07/10/2006

Ambiente de Desenvolvimento

Histórico da Revisão

Data	Versão	Descrição	Autor
07/10/2006	1.0	Versão inicial	Lidimon Cristiano - Newton Campos

1. Introdução

1.1 Finalidade

Este documento apresenta as ferramentas que serão utilizadas durante o projeto.

1.2 Escopo

Este documento aplica-se na fase de iniciação, elaboração, construção e transição.

1.3 Definições, Acrônimos e Abreviações.

Consulte documento Glossário.

1.4 Referências

Nenhuma

2. Visão Geral

O restante deste documento descreve as ferramentas a serem utilizadas a partir da fase de elaboração.

3 Ambiente Fase de Elaboração

Tipo de Artefato	**Ferramentas**
Artefatos Formais (documento visão, lista de riscos etc).	Microsoft Word
Diagramas UML	Plugin Omondo para Eclipse 3.1 - Rational Rose
Modelo de Dados	Erwin

4 Ambiente Fase de Construção

4.1 Linguagem

Java JVM: jre-1-5-0-10-windows-i586-p

4.2 Ferramenta de Desenvolvimento

Eclipse 3.1 com Interface Swing

4.3 Banco de Dados

Firebird - Versão 2.0.0.12748

Apêndice J

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Elaboracao/Disciplinas/9.Gerenciamento de Projeto	Data: 07/10/2006

Plano de Iteração

Histórico da Revisão

Data	Versão	Descrição	Autor
07/10/2006	1.0	Versão inicial	Lidimon Cristiano

1. Introdução

1.1 Finalidade

Este Plano de Iteração descreve os planos para a Iteração da fase de Elaboração. Durante essa iteração, será definido o Documento de Arquitetura do Software.

1.2 Escopo

Este plano de Iteração possui foco nas disciplinas de Requisitos, Gerenciamento de Projeto, Ambiente e principalmente na disciplina de Análise e Design.

1.3 Definições, Acrônimos e Abreviações.

Consulte documento Glossário.

1.4 Referências

Especificação de Casos de Uso

2. Plano

2.1 Tarefas da Iteração

A tabela a seguir ilustra as tarefas a serem realizadas com as respectivas datas de início e término.

Tarefa	**Disciplina**	**Início**	**Término**	**Papel responsável**
Elaboração		07/10/06	25/10/06	
Definir Ambiente de Desenvolvimento	Ambiente	07/10/06	08/10/06	Lidimon
Elaborar as Classes referentes aos Casos de Uso	Análise e Design	09/10/06	10/10/06	Lidimon
Elaborar Diagrama de Seqüência	Análise e Design	11/10/06	12/10/06	Lidimon - Newton
Revisar e refinar Diagrama de Classes	Análise e Design	13/10/06	14/10/06	Lidimon
Definir Modelo de Dados	Análise e Design	15/10/06	16/10/06	Lidimon
Elaborar Documento de Arquitetura de Software	Análise e Design	17/10/06	20/10/06	Lidimon - Newton
Revisar Plano de Desenvolvimento de Software e Lista de Riscos.	Gerenciamento de Projeto	22/10/06	25/10/06	Lidimon - Newton

Os seguintes artefatos serão gerados ou revisados durante a Iteração desta fase.

Disciplina	**Artefato liberado durante a disciplina**	**Proprietário responsável**
Requisitos	Casos de Uso	Lidimon
Gerenciamento de Projeto	Plano de Desenvolvimento de Software, Lista de Riscos	Lidimon - Newton
Análise e Design	Documento de Arquitetura de Software	Lidimon - Newton
Análise e Design	Modelo de Design	Lidimon
Análise e Design	Modelo de Dados	Lidimon
Ambiente	Ambiente de Desenvolvimento	Lidimon

3 Casos de Uso Os casos de uso já foram especificados no artefato de Especificações de Casos de Uso na fase de Iniciação.

4 Critérios de Avaliação

A meta principal desta iteração é desenvolver o Modelo de Design, Modelo de Dados e Arquitetura do Software.

Apêndice K

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Elaboracao/Disciplinas/9.Análise e Design	Data: 17/10/2006

Arquitetura de Software

Histórico da Revisão

Data	Versão	Descrição	Autor
17/10/2006	1.0	Versão inicial	Lidimon Cristiano

1. Introdução

1.1 Finalidade

Este documento fornece uma visão geral da arquitetura do sistema.

1.2 Escopo

Este documento se aplica ao SVCE que será desenvolvido pela integração de contexto (digrama de classes, modelo de dados etc).

1.3 Definições, Acrônimos e Abreviações.

Consulte documento Glossário.

1.4 Referências

Documento Visão

Plano de Iteração

Especificação de Casos de Uso

Plano de Desenvolvimento de Software

2. Representação da Arquitetura

Este documento apresenta a arquitetura com uma série de visões: visão de casos de uso, visão de implantação e visão de implementação. Essas visões são em UML.

3. Metas e Restrições da Arquitetura

Nenhuma

4. Visão de Casos de Uso

A Visão de Casos de Uso é uma entrada importante para a seleção do conjunto de cenários e/ou casos de uso que são o foco de uma iteração. Ela descreve o conjunto de cenários e/ou os casos de uso que representam alguma funcionalidade central e significativa.

Os casos de uso deste sistema estão listados a seguir.

- Registrar Venda

- Cadastrar Funcionário

- Controlar Estoque

- Cadastrar Forma de Pagamento

- Cadastrar Produto - Cadastrar Fornecedor

- Consultar Histórico/Imprimir

O diagrama a seguir representa os casos de uso do sistema.

5. Descrição dos Casos de Uso

Consulte artefato de Especificação de Casos de Uso.

6. Visão Geral

Descreve as classes mais importantes, sua organização em pacotes e subsistemas de serviço e a organização desses subsistemas em camadas. A visão lógica do SCVE é composta por dois pacotes.

Apresentação: contém classes para cada um dos formulários que os atores utilizam para se comunicar com o sistema. Trata-se basicamente do aplicativo.

Consultas: contém classes de consultas.

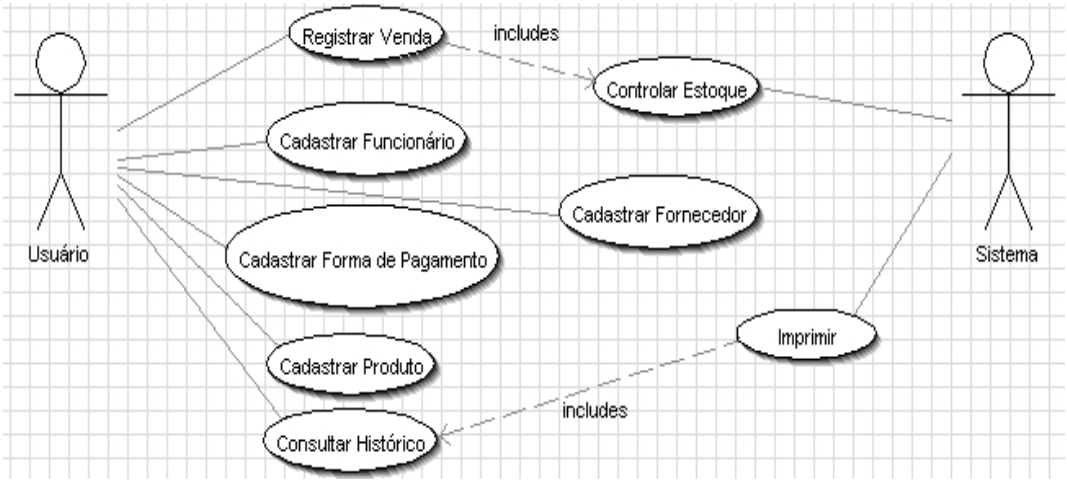

Figura 7.5: Visão dos Casos de Uso

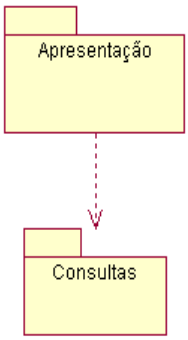

Figura 7.6: Visão Geral: Pacotes do sistema

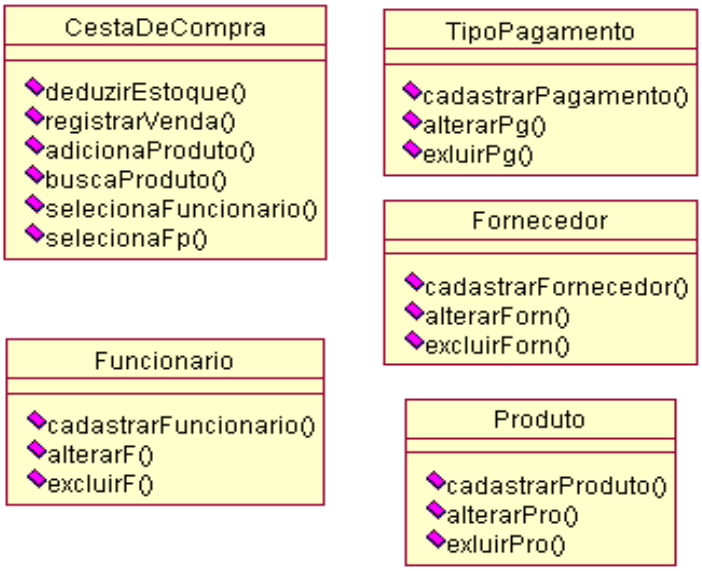

Figura 7.7: Visão Lógica: Pacote de Apresentação

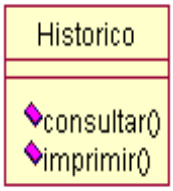

Figura 7.8: Visão Lógica: Pacote de Consulta

6.1 Visão Lógica

Consulte Diagrama de Classes Final

Referência: D:/ProjetoSCVE/Elaboracao/Disciplinas/1.Análise e Design

7. Pacote de Apresentação

8. Pacote de Consulta

9. Visão de Implantação

Esta seção descreve a configuração (hardware) onde o software será implantado e executado.

A máquina SCVE é qualquer computador com JVM.

10. Visão de Implementação

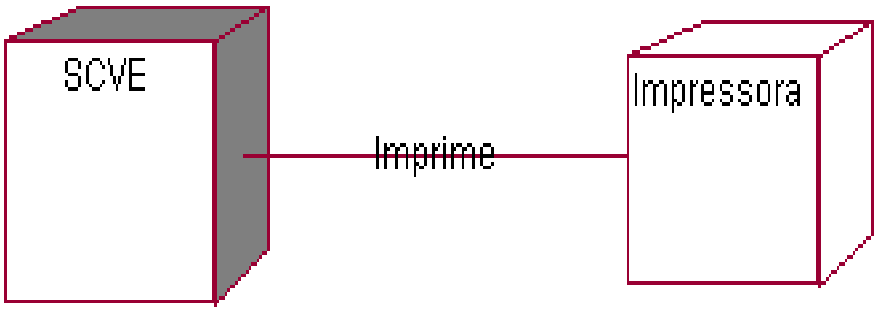

Figura 7.9: Visão de Implantação

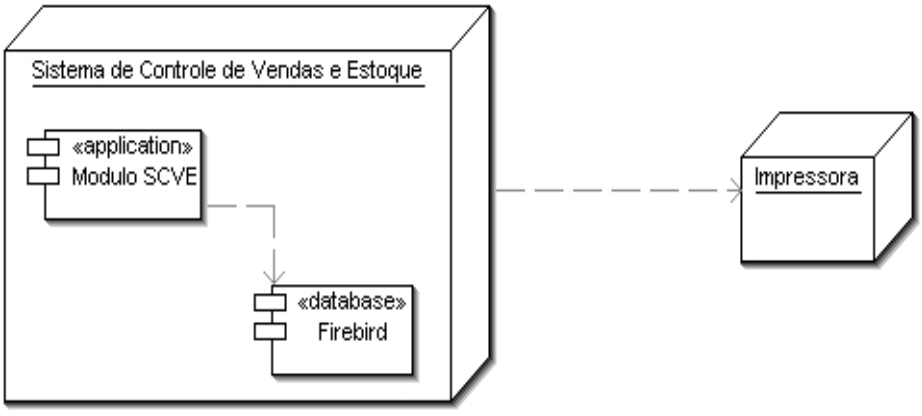

Figura 7.10: Visão de Implementação

Apêndice L

Sistema de Controle de Vendas e Estoque	Versão: 1.0
D:/ProjetoSCVE/Elaboracao/Disciplinas/9.Análise e Design	Data: 17/10/2006

Plano de Desenvolvimento de Software

Histórico da Revisão

Data	Versão	Descrição	Autor
02/10/2006	1.0	Versão inicial	Lidimon Cristiano / Newton Campos
07/10/2006	2.0	Atualização do Plano para a fase de Elaboração.	Lidimon Cristiano / Newton Campos

1. Introdução

1.1 Finalidade

A finalidade deste Plano de Desenvolvimento de Software é definir as atividades de desenvolvimento no que diz respeito às fases e iterações necessárias para a implementação do Sistema de Controle de Vendas e Estoque (SCVE) para a Locadora de Games.

1.2 Escopo

Este Plano de Desenvolvimento de Software descreve o plano geral a ser usado pela equipe para desenvolver o sistema. Os detalhes de iterações individuais serão descritos nos Planos de Iteração.

1.3 Definições, Acrônimos e Abreviações

Consulte documento Glossário.

1.4 Referências

Documento Visão

Lista de Riscos

Planos de Iteração

2. Visão Geral do Projeto

2.1 Finalidade, escopo e objetivos do projeto

Este projeto implementará um Sistema de Controle de Vendas e Estoque para substituir o antigo controle de vendas, que é realizado através de planilhas eletrônicas. Consulte Documento Visão para maiores detalhes.

2.2. Suposições e Restrições

Nenhuma

2.3. Produtos Liberados do Projeto

Os produtos liberados a seguir serão produzidos ou atualizados durante o projeto: Documento Visão

Plano de Desenvolvimento de Software

Caso de Negócio

Glossário

Especificação de Casos de Uso

Plano de Iteração

Lista de Riscos

Documento de Arquitetura de Software

2.4 Evolução do Plano de Desenvolvimento de Software

O Plano de Desenvolvimento de Software será revisado antes do início de cada fase de iteração. As datas previstas para o final de cada fase esta prevista a seguir.

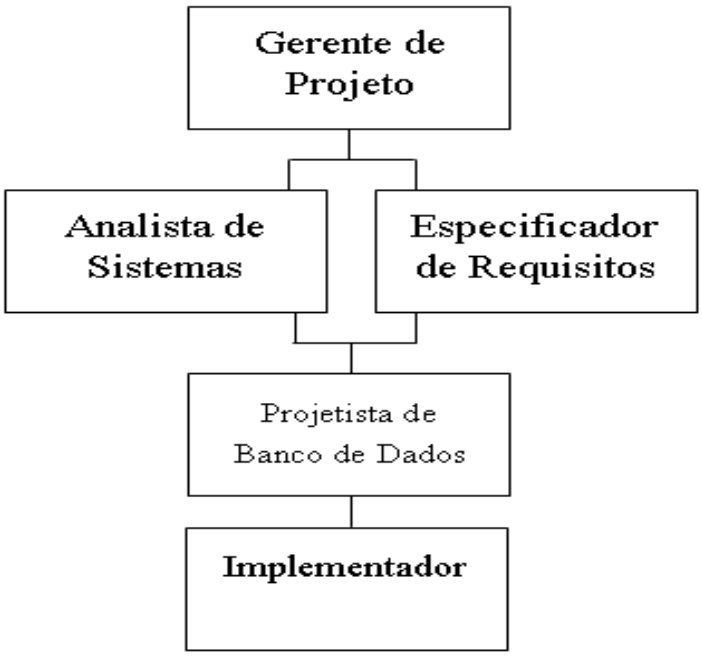

Figura 7.11: Estrutura Organizacional Versão 2.0.

Fase	**Término**
Concepção	06/10/06
Elaboração	25/10/06.

3 Organização do Projeto

3.1 Estrutura Organizacional

A equipe de projeto da fase de iniciação e elaboração será organizada da seguinte maneira:

3.2 Interfaces Externas

O Gerente da Locadora de Games e os funcionários mais experientes irão informar os requisitos do sistema. O gerente irá testar as funcionalidades do sistema.

3.3 Papéis e Responsabilidades

A tabela a seguir mostra as funções representadas no diagrama anterior e suas respectivas responsabilidades.

Papel	**Responsabilidade**
Gerente de Projeto	O Gerente de Projeto aloca recursos, especifica prioridades, coordena as interações com os clientes e usuários e, geralmente, tenta manter a equipe de projeto centrada na meta correta. O Gerente de Projeto também estabelece um conjunto de práticas que garante a integridade e qualidade dos artefatos do projeto.
Analista de Sistemas	O papel analista de sistemas lidera é coordenar a identificação de requisitos e a modelagem de casos de uso, delimitando o sistema e definindo sua funcionalidade; por exemplo, estabelecendo quais são os atores e casos de uso existentes e como eles interagem.
Especificador de Requisitos	O papel especificador de requisitos detalha a especificação de uma parte da funcionalidade do sistema, descrevendo o aspecto Requisitos de um ou de vários casos de uso e outros requisitos de software de apoio.
Projetista (Design) de Banco de Dados	O papel designer de banco de dados é definir tabelas, índices, visões, restrições, triggers, procedimentos armazenados, parâmetros de armazenamento e outras construções específicas de um banco de dados necessárias para armazenar, recuperar e excluir objetos persistentes.
Implementador	O papel implementador é responsável por desenvolver e testar componentes de acordo com os padrões adotados para o projeto.

4.1 Estimativas do Projeto

A fase de iniciação deste projeto deve durar uma semana.

A fase de elaboração deve durar duas semanas.

Treinamento para fase de construção deve durar 15 dias. Consulte lista de Riscos: Item: Plano de Contingência.

A fase de construção deve durar 20 dias.

A fase de transição deve durar uma semana.

4.2 Plano do Projeto

As fases do projeto podem ser visualizadas da seguinte maneira:

Fase	**Início**	**Fim**
Concepção	01/10/06	06/10/06
Elaboração	07/10/06	25/10/06
Construção	05/01/07	25/01/07
Transição	26/01/07	03/02/07

4.2.1 Plano de Fase

O desenvolvimento será conduzido usando uma abordagem com fases onde várias iterações ocorrem dentro de uma fase. As fases e a linha de tempo relativa são mostradas na tabela a seguir:

Fase	**Nº de Iterações**	**Início**	**Fim**
Fase de Iniciação	1	01/10/06	06/10/06
Fase de Elaboração	1	09/10/06	25/10/06

Os marcos que indicam o fim de cada fase podem ser vistos na tabela a seguir.

Descrição	Marco
Fase de Cencepção	A fase de concepção desenvolverá os requisitos do produto e estabelecerá o caso de negócio. Os principais casos de uso serão desenvolvidos, bem como este Plano de Desenvolvimento de Software.
Fase de Elaboração	A Fase de Elaboração analisará os requisitos e desenvolverá o protótipo de arquitetura. Após concluir a Fase de Elaboração, o projeto já estará concluído e pronto para a fase de construção no qual poderá ser implementado.

4.2.2 Objetivos da Iteração

Fase	Nº de Iterações	Descrição	Marcos Associados
Concepção	Iteração Preliminar	Definir os principais requisitos do sistema, caso de negócio, iteração e o plano de desenvolvimento de software.	Aprovação do Caso de Negócio Identificação dos Riscos
Elaboração	Iteração 1	Define as classes do sistema, modelo de dados e arquitetura.	Documento de Arquitetura de Software.

4.2.3 Releases

Neste ponto nenhum release será apresentado.

4.2.4 Orçamento

Este projeto não possui orçamento, a experiência adquirida pela equipe servirá como forma

de pagamento pelos esforços realizados.

Referências Bibliográficas

[1] Ibm rational software. Technical report, IBM, Acesso em: setembro de 2006. Disponível em: http://www.rational.com.

[2] KRUCHTEN , Philippe. *Introdução ao RUP Rational Unified Process*. ISBN: 85-7393-275-9. Ciência Moderna, 2001.

[3] PRESSMAN , Roger S. Engenharia de *Software*. ISBN: 85-8680-425-8. McGraw-Hill, 2002.

[4] TELES , Vinicius Manhaes. *Extreme Programming*. ISBN: 85-7522-047-0. Novatec, 2004.

[5] Shari Lawrence , PFLEEGER. *Engenharia de Software: Teoria e Prática 2ª Edição*. ISBN: 85-879-1831-1. Prentice Hall, 2004.

[6] Extreme programming. Technical report, Improveit, Acesso em: setembro de 2006. Disponível em: http://www.improveit.com.br.

[7] James RUMBAUGH Ivar JACOBSON Grady, BOOCH. *UML: Guia do Usuário*. ISBN: 85-352-0562-4. Elsevier, 2000.

[8] Gileanes T. A. , GUEDES. *UML Uma Abordagem Prática*. ISBN: 85-7522-052-7. Novatec, 2004.

[9] Ernani MEDEIROS. *Desenvolvendo Software com UML 2.0 Definitivo*. ISBN: 85-346-1529-2. Makron Books, 2004.

[10] Michael BOGGS, Wendy / BOGGS. *Mastering UML com Rational Rose 2002 A Bíblia.* ISBN: 85-887-4543-7. Alta Books, 2002.