Universidade Federal de Santa Catarina
Programa de Pós-Graduação em
Engenharia de Produção

Cleber Aparecido Vidotti

METODOLOGIA SIMPLIFICADA DE DESENVOLVIMENTO DE SOFTWARE PARA EMPRESAS DE PEQUENO E MÉDIO PORTE: UMA APLICAÇÃO PRÁTICA NA WOPM INFORMÁTICA

Dissertação de Mestrado

Florianópolis
2003

Cleber Aparecido Vidotti

METODOLOGIA SIMPLIFICADA DE DESENVOLVIMENTO DE SOFTWARE PARA EMPRESAS DE PEQUENO E MÉDIO PORTE: UMA APLICAÇÃO PRÁTICA NA WOPM INFORMÁTICA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina como requisito parcial para obtenção do grau de Mestre em Engenharia de Produção.

Orientador: Prof. José Leomar Todesco, Dr

Florianópolis

2003

Cleber Aparecido Vidotti

METODOLOGIA SIMPLIFICADA DE DESENVOLVIMENTO DE SOFTWARE PARA EMPRESAS DE PEQUENO E MÉDIO PORTE: UMA APLICAÇÃO PRÁTICA NA WOPM INFORMÁTICA

Esta dissertação foi julgada e aprovada para a
Obtenção do grau de **Mestre em Engenharia de Produção** no **Programa de Pós-Graduação em Engenharia de Produção** da
Universidade Federal de Santa Catarina

Florianópolis, 04 de abril de 2003

Prof. Edson Pacheco Paladini, Dr.
Coordenador do Programa

BANCA EXAMINADORA

_____

Prof. José Leomar Todesco, Dr.
Universidade Federal de Santa Catarina
Orientador

_____

Prof. Aran Bey Tcholakian Morales, Dr.
Universidade Federal de Santa Catarina

_____

Prof. Oscar Ciro Lopes Vaca, Dr.
Universidade Federal de Santa Catarina

A minha esposa pelo
apoio constante e a
toda minha família por
apostarem em mim.

Agradecimentos

À Universidade Federal de Santa Catarina.

À Universidade Estadual do Oeste do Paraná.

Ao orientador Prof. Dr. José Leomar Todesco, meu orientador, que sabiamente me aconselhou e conduziu de forma pontual e competente.

À W.O.PM. Informática pela oportunidade.

E a todos os que direta ou indiretamente contribuíram para a realização desta pesquisa.

RESUMO

VIDOTTI, Cleber. **Metodologia Simplificada de Desenvolvimento de Software para Empresas de Pequeno e Médio Porte: Uma Aplicação Prática na WOPM Informática.** Florianópolis, 2003, 106 f. Dissertação(Mestrado em Engenharia de Produção), PPGEP, UFSC.

A qualidade do software está fortemente relacionada à qualidade do processo de software. Os problemas do desenvolvimento e da manutenção de software não podem ser solucionados apenas com a introdução de ferramentas e ambientes de desenvolvimento de software. O entendimento de que a existência de um método de desenvolvimento de software simplificado poderia reduzir tempo e custo de desenvolvimento, bem como uma melhoria significativa na qualidade do software desenvolvido, é o objeto desta pesquisa. Fez-se uma contraposição às várias metodologias de desenvolvimento de software existentes, elegendo o método FUSION como a metodologia a ser aprofundada. Partindo deste método procurou-se detalhar as atividades de análise, buscando a simplificação, para que atendesse as necessidades e porte da empresa estudada. As atividades propostas foram aplicadas na empresa e monitoradas durante a execução de um projeto. Concluiu-se que A metodologia simplificada proposta foi assimilada pela equipe rapidamente e trouxe benefícios para a empresa como a entrega do software no prazo e uma maior percepção, por parte do cliente, da qualidade do mesmo.

Palavras-chave: Metodologia, Processo, Software, Desenvolvimento.

ABSTRACT

VIDOTTI, Cleber. **Metodologia Simplificada de Desenvolvimento de Software para Empresas de Pequeno e Médio Porte: Uma Aplicação Prática na WOPM Informática.** Florianópolis, 2003, 106 f. Dissertação(Mestrado em Engenharia de Produção), PPGEP, UFSC.

The quality of software is strongly related to the quality of software process. The problems of software development and maintenance cannot be solved only by introduction of tools and environments of software development. The understanding that the existence of a simplified method of software development could reduce the time and cost of development, as well as a significative advance in quality of the developed software, is it's the object of this search. A comparison to several methodologies of existent software development was done, choosing the method FUSION as the methodology to be deepen. Leaving from this method we sought to detail the activities of analysis, seeking the simplification, so that it could attend to the necessities and size of company studied. The proposed activities were applied in the company and monitored during the execution of the project. Thus, we concluded that the proposed simplified methodology was assimilated by the team quickly and brought benefits to company like the delivery of the software on time and a greater perception by the client, of the software quality.

Key-words: Methodology, Process, Software, Development.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

1 INTRODUÇÃO

1.1 Contextualização e Tema da Pesquisa

Atualmente, uma grande parte da população mundial depende de aplicações de software para realizar suas atividades diárias. Segundo REED (2000), se alguns sistemas de uso global deixarem de funcionar, aproximadamente 40% da população mundial sofrerá as conseqüências do problema.

Fazer software não é uma tarefa fácil. Fazer software de qualidade é mais difícil. Os resultados obtidos ao logo do tempo nos mostram padrões de baixa qualidade e de custos e prazos completamente ultrapassados.

Desde 1968 e 1969 fala-se em metodologias de desenvolvimento de software, mas resta muito para ser feito na busca de qualidade e produtividade no desenvolvimento de software.

Ainda hoje, qualidade do software é tema muito discutido. Temos mercados cada vez mais competitivos e clientes cada vez mais exigentes, fazendo com que as empresas busquem desenvolver produtos cada vez melhores. Fuggetta (2000) afirma que a qualidade do software está fortemente relacionada à qualidade do processo de software.

As empresas buscam constantemente o melhoramento de seus processos, exigindo profissionais qualificados para desenvolver softwares de qualidade. Assim, vale a pena analisar os diversos esforços que foram efetuados pelas empresas ao longo do tempo, e perceber por que alguns processos não foram efetivos na resolução dos problemas. As organizações que forem capazes de integrar, harmonizar e acelerar os seus processos de desenvolvimento e também de manutenção de software terão a primazia do mercado.

Segundo Lindvall (2000), a preocupação com o processo de software está relacionada à necessidade de entender, avaliar, controlar, aprender, comunicar, melhorar, predizer e certificar o trabalho dos engenheiros de software. Para isso é

preciso documentar, definir, medir, analisar, avaliar, comparar e alterar os processos.

Para Fuggetta (2000), a pesquisa em processo de software trata dos métodos e tecnologias utilizados para avaliar, apoiar e melhorar as atividades de desenvolvimento e manutenção de software. A primeira contribuição importante da pesquisa na área de processo de software é: o convencimento de que desenvolver software é fruto de um esforço coletivo, complexo e criativo e de que a qualidade do software depende das pessoas, da organização e dos procedimentos usados em seu desenvolvimento. Os problemas do desenvolvimento e da manutenção de software não podem ser solucionados com a introdução de ferramentas e ambientes de desenvolvimento de software, embora estes sejam aspectos importantes. Também não podem ser solucionados com a seleção de um modelo de ciclo de vida, mesmo que este seja adequado ao desenvolvimento. A definição e o uso de processo de software envolve a complexa inter-relação de fatores organizacionais, culturais, tecnológicos e econômicos.

Questões como porte da empresa e a cultura organizacional, objetivos de projetos específicos, recursos disponíveis, tecnologias de desenvolvimento, conhecimento e experiência da equipe impõem características aos processos. Assim, as situações é que definem como devem ser os processos de software.

Neste trabalho, através de um estudo de caso, avaliaremos os processo de desenvolvimento de software da WOPM Informática, que segundo a tabela 1.1 fornecida pelo Sebrae (2003), é uma empresa de pequeno porte, que se caracteriza pela utilização de um processo de desenvolvimento de software baseada na experiência da empresa e sem nenhuma fundamentação metodológica. Esta avaliação será baseada nos processos de desenvolvimento estabelecido nos projetos executados pela empresa, que terá como foco, aspectos como tempos e métodos.

Quadro 1.1 - Classificação De Empresas Por Número De Empregados

PORTE	Empregados
Microempresa	No comércio e serviços até 09 empregados Na indústria até 19 empregados
Pequeno Porte	No comércio e serviços de 10 a 49 empregados Na indústria de 20 a 99 empregados
Médio Porte	No comércio e serviços de 50 a 99 empregados Na indústria de 100 a 499 empregados
Grande Porte	No comércio e serviços mais de 99 empregados Na indústria mais de 499 empregados

Fonte: Adaptado do site do SEBRAE(2003)

Será feita uma contraposição às várias metodologias de desenvolvimento de software existentes, onde serão colocadas as vantagens e desvantagens do processo atual, da adoção de uma nova metodologia ou da modificação do processo atual.

Como produto final deste trabalho, teremos uma proposta de melhoria ou alteração do processo de desenvolvimento existente e uma aplicação prática desta proposta.

1.2 Problema

Mais do que nunca, as empresas de desenvolvimento de softwares, nesse início de século, buscarão formas de aumentarem a produtividade, visando a qualidade produto e o cumprimento das metas estabelecidas com o cliente. Essa realidade atinge desde as pequenas empresas até as mega-corporações.

Diante dessa expectativa, com base na hipótese de que a metodologia de desenvolvimento de software adotada interfere diretamente na performance do processo, e logo, na competitividade da empresa, procurou-se estudar e responder ao problema da pesquisa, formulado através da seguinte pergunta: **existe uma**

metodologia[1] de desenvolvimento de software simplificado e detalhado em atividades, que atenda de modo eficiente à produção de software, podendo ser implantado na forma de uma fábrica de software?

1.3 Hipóteses da Pesquisa

Face à questão formulada, a existência de uma metodologia simplificada de desenvolvimento de software é de fundamental importância para que empresas de desenvolvimento possam cumprir suas metas, aumentando a qualidade do produto e a satisfação do cliente.

A existência de uma metodologia de desenvolvimento de software poderia reduzir tempo e custo de desenvolvimento, bem como uma melhoria significativa na qualidade do software desenvolvido.

1.4 Objetivos

1.4.1 Objetivo Geral

O objetivo é propor uma metodologia de desenvolvimento de software simplificado e flexível, onde as atividades de cada fase estejam detalhadas, permitindo assim que o mesmo possa ser implantado na forma de um modelo de fábrica de software, buscando a melhoria do processo de produção de software e o aumento da qualidade do software produzido.

---

[1] A metodologia é o estudo da melhor maneira de, num determinado estado de conhecimentos, abordar determinados problemas. Ela não procura soluções, mas contribui na escolha das maneiras de encontrá-las, integrando os conhecimentos adquiridos sobre os métodos em vigor nas diferentes disciplinas científicas ou filosóficas.

1.4.2 Objetivos Específicos

Tendo em vista o objetivo geral do trabalho, traçaram-se os seguintes objetivos específicos:

1. Identificar na fundamentação teórica, a metodologia de desenvolvimento de software que melhor se adapta ao modelo de fábrica de software;
2. Analisar a metodologia escolhida e identificar as atividades necessárias às fases de desenvolvimento;
3. Mapear e descrever as atividades da metodologia;
4. Identificar qual a fase crítica do processo de desenvolvimento da empresa onde será aplicado a metodologia proposta;
5. Implantar as atividades da metodologia simplificada na fase identificada como crítica;
6. Analisar os resultados.

1.5 Justificativa

Segundo Coleman (1996, pg.3), os problemas relacionados ao desenvolvimento de software são caracterizados, freqüentemente, pela baixa capacidade de previsão, pela baixa qualidade dos softwares, pelos altos custos de manutenção e pela duplicação de esforços.

Os responsáveis pelo desenvolvimento têm sérios problemas na previsão do tempo e do esforço necessários para produzir um sistema que satisfaça os requisitos do usuário. Segundo Brooks (1982), medidas quantitativas de tempo e esforço não são facilmente convertidas entre si. Os erros são encontrados apenas nas etapas finais do processo de codificação, fruto de uma análise falha combinada com a pressa para a entrega do software. Tanto a manutenção corretiva quanto a evolutiva são dispendiosas quando o sistema não foi construído com uma arquitetura clara e

visível. Os projetos são freqüentemente iniciados “da estaca zero”, compartilhando pouco código de projetos anteriores. Mesmo no âmbito de um mesmo projeto esta situação pode ser encontrada.

Para a solucionar estes problemas pode-se tomar dois caminhos: aperfeiçoar as ferramentas utilizadas pela equipe que executa o desenvolvimento, o que nem sempre traz resultados satisfatórios, mesmo porque desenvolver software é um problema de métodos e técnica, em suma, de atividade humana. Assim a outra alternativo é aperfeiçoar o processo utilizado. É claro que, quando ferramentas e processo são adequadamente combinados, obtêm-se melhores resultados.

Muitas empresas de desenvolvimento de software passam anos trabalhando, sem perceber os ganhos de qualidade, tempo e dinheiro que haveria com o estabelecimento de uma metodologia de trabalho, seja ela estruturada ou orientada a objetos.

Muitas empresas buscam implantar uma metodologia de desenvolvimento, mas encontram dificuldades em estabelecer e padronizar sozinhas, as atividades que compõem cada fase da metodologia escolhida, fazendo com que, freqüentemente, não consigam atingir os objetivos propostos inicialmente, ou até mesmo desistam da implantação, face à necessidade de contratação de consultorias caras. Particularmente, são as empresas de pequeno e médio porte que sentem mais estes efeitos, pois a falta de recursos aliada a “pressa” dos clientes em receberem o produto, não permite a implantação e execução das atividades formais que cada metodologia exige.

Outras empresas utilizam o conceito de fábrica de software como forma de marketing, sem que realmente tenham um processo de desenvolvimento adequado a esta realidade.

Portanto, nesse sentido, é importante estabelecer uma metodologia simplificada e flexível, com atividades detalhadas e que seja de fácil implantação, buscando melhorar o processo de desenvolvimento das empresas e a qualidade do software produzido.

1.6 Desenvolvimento do Trabalho

Para desenvolver este trabalho, alguns procedimentos preliminares foram utilizados. A figura 1, a seguir apresenta sucintamente o roteiro dos procedimentos utilizados no trabalho.

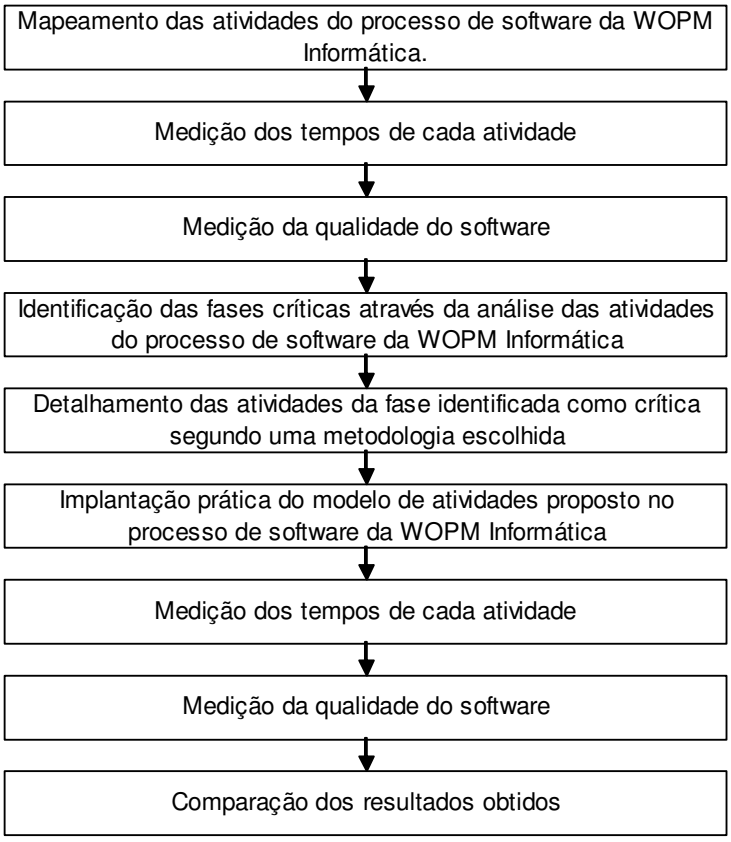

Figura 1.1: Roteiro dos procedimentos.

Na 1ª etapa, procurou-se mapear todas as atividades de processo de desenvolvimento de software da WOPM Informática, nas fases de projeto e implementação.

Na 2ª etapa, foram feitas medições dos tempos gastos com cada atividade do processo, buscando estabelecer parâmetros para comparações futuras.

Na 3ª etapa foi efetuada uma parametrização da qualidade do software produzido pelo processo atual, principalmente sob os aspectos: atendimento dos requisitos do cliente, quantidade de erros, cumprimento de prazos e custo/valor do software.

Na 4ª etapa foi feita uma análise das atividades do processo atual, buscando identificar qual fase teve maior impacto no software produzido.

Na 5ª etapa procurou-se estabelecer, através de uma análise teórica, qual metodologia de desenvolvimento melhor se adequava ao processo da empresa, sendo feito em seguida, o detalhando das suas fases em atividades.

Na 6ª etapa, o processo proposto foi implantado na fase de projeto.

Na 7ª e 8ª etapa, foram realizadas as mesmas medições realizadas no processo anterior e no software produzido.

Por fim, na 9ª etapa é feita uma comparação dos resultados obtidos, buscando fazer uma análise dos pontos positivos e negativos do processo proposto.

1.7 Limitações do Estudo

O principal entrave que condicionou a limitação deste estudo refere-se à insipiente bibliografia a respeito do detalhamento das atividades descritas por cada metodologia, bem como, não foi possível localizar nenhum trabalho, em nível de mestrado ou doutorado, sobre a temática estudada.

Outro aspecto a considerar são os resultados das medições feitas nas atividades e nos softwares que, por se tratar de dados quantitativos, às vezes, distorcem os resultados de uma pesquisa ou estudo, podendo generalizar os resultados sem uma análise adequada da situação.

1.8 Estrutura do Trabalho

A dissertação foi estruturada em capítulos onde serão apresentados os seguintes temas:

No capítulo 2, foi feita a Revisão da Literatura, referenciando os trabalhos escritos sobre metodologias de desenvolvimento mais conhecidas e utilizadas.

No capítulo 3, é dado um enfoque especial na metodologia Fusion, que é a metodologia base do nosso estudo.

No capítulo 4, apresenta-se uma proposta de processo de desenvolvimento de software simplificado, baseado no método Fusion, com um detalhamento de atividades para que não haja dúvidas quanto a sua implantação.

No capítulo 5 foram detalhadas as atividades do processo de desenvolvimento da WOPM Informática, apresentado o resultado das medições dos tempos gastos com cada atividade e estabelecido a percepção de qualidade do software produzido com este processo. Também é feita a validação da proposta, através de uma aplicação prática da proposta apresentada. Nesta aplicação, a fase de análise do processo de desenvolvimento da WOPM Informática foi escolhida. Ainda neste capítulo é apresentado o resultado das medições dos tempos das atividades e da qualidade do software produzido depois da implantação.

Finalmente, no capítulo 6, são descritas as conclusões e recomendações finais acerca do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

A abordagem orientada a objetos surgiu inicialmente com as pesquisas e o desenvolvimento de linguagens, como Smalltalk, em que os objetos são o ponto focal da programação. Objetos são agrupados em classes que definem de forma precisa à mesma interface para todas as suas instâncias (objetos) e as classes são os blocos básico de construção das linguagens orientadas a objetos. Os objetos são entidades dinâmicas e quando o sistema está em execução trocam mensagens entre si. As mensagens invocam métodos que executam as atividades relevantes do sistema. Os objetos que enviam uma mensagem não precisam conhecer os detalhes da implementação interna do objeto ao qual a mensagem foi enviada, apenas a interface com a qual ele se relaciona com seus clientes e as atividades que ele executa.

A abordagem orientada a objetos procura diminuir a distância entre os conceitos da aplicação no mundo real e a implementação através de uma linguagem de programação. O mesmo conceito de objeto é usado ao longo de todas as fases do desenvolvimento, em diferentes níveis de abstração. Existem hoje várias propostas de métodos orientados a objetos para desenvolvimento de software. Alguns deles, como OMT, citado por Rumbaugh (1991), evoluíram de métodos tradicionais como a análise estruturada.

O número de metodologias propostas para o desenvolvimento de software atingiu um número demasiado elevado, o que torna virtualmente impossível a sua apresentação. Por isso, enumeramos algumas metodologias, estruturadas e orientadas a objeto, conhecidas que maior relevância e divulgação tiveram. Para além destas, existiram outras contribuições importantes que não estão incluídas aqui por não apresentarem uma perspectiva integrada de todo o processo de desenvolvimento, mas apenas sugerirem anotações ou técnicas de modelagem. É o caso, por exemplo, das propostas de Tom DeMarco (1978) e de Meiler Page-Jones (1980).

A metodologia mais divulgada e que alcançou maior sucesso foi o SSADM (Structured Systems Analysis and Design Methodology), proposta em 1981 por Learmonth, e alvo de sucessivas revisões, a última das quais em meados da década

de 90, com o aparecimento da versão 4+ conforme Weaver (1998). Durante muito tempo (e ainda hoje) foi considerada a metodologia de referência e ensinada em diversos cursos universitários. No Reino Unido, foi durante muito tempo obrigatória a sua utilização em todos os projetos relacionados com o desenvolvimento de sistemas de informação a nível governamental.

O Stradis (Structured analysis, design and implementation of information systems) foi uma metodologia desenvolvida por Gane e Sarson em finais da década de 70 segundo Gane (1982), baseada na filosofia de decomposição funcional top-down, e suportada essencialmente pela utilização de diagramas de fluxos de dados.

Já a Yourdon Systems Method, é uma metodologia proposta por Edward Yourdon, revista em 1993 e descrita com algumas atualizações em 1999 pelo próprio Yourdon (1999); é semelhante a Stradis, pois recorre muito a decomposição funcional, mas também atribui uma importância significativa à estrutura dos dados.

Também tivemos a “Engenharia de Informação”, proposta por James Martin (1989), integra muitos dos conceitos, melhores práticas, modelos e técnicas das metodologias estruturadas e do desenvolvimento de software dos anos 80 em uma abordagem coerente a todas as atividades do processo de desenvolvimento de software. As suas raízes estão no trabalho originalmente desenvolvido pela IBM na sua metodologia Business Systems Planning. Ao contrário das outras, a Engenharia da Informação tem uma preocupação estratégica com a definição dos sistemas de informação, o que é expresso nos diferentes estágios de evolução do processo designadamente: (1) planejamento estratégico dos sistemas de informação; (2) análises de áreas do negócio; (3) desenho dos sistemas e (4) implementação.

O método Booch foi proposto por Grady Booch em 1991 (cujo livro de referência teve uma segunda edição em 1994, Booch (1994)), e baseia-se na idéia da repetição de atividades de um processo de desenvolvimento de modo a refinar o modelo em sucessivas iterações. As suas principais atividades estão orientadas para a identificação de classes e objetos e respectivas características e para a determinação das relações entre classes.

O OMT foi proposto por James Rumbaugh em 1991, que concentrou a sua proposta na análise e desenho de software, às quais aplicou técnicas orientadas por

objetos. A sua metodologia apresentava essencialmente três modelos principais: estático ou de objetos (onde se representavam classes, objetos, a hierarquia e outras relações), dinâmico (apresentava o comportamento dos objetos e do sistema global) e funcional (diagrama de fluxo de informação no sistema semelhante aos diagramas de fluxos de dados).

O OOSE foi proposto por Ivar Jacobson em 1992, resulta da evolução do modelo Objectory (também do mesmo autor) e a sua maior contribuição foi à introdução da noção de caso de utilização que funciona como uma descrição da interação entre o utilizador e o sistema.

O OOAD foi proposto por Peter Coad e Edward Yourdon em 1991, apresentava como uma das suas principais vantagens o fato de ser muito simples (ao nível dos conceitos, atividades e diagramas) o que o tornava um dos mais fáceis de compreender. As suas principais atividades relacionadas com a análise são no fundo aquilo que todos nós esperaríamos realizar num processo que aplicasse as noções da orientação por objetos: Identificar objetos utilizando critérios simples (substantivos), definir uma estrutura de relações generalização-especificação, definir uma estrutura de relações de associação (whole-part), identificar assuntos (subsistemas), definir os atributos e definir os serviços.

A metodologia de Wirfs-Brock não efetua uma distinção clara entre análise e desenho, e a sua principal contribuição foi à definição de um diagrama designado por CRC cards (Class-Responsibility-Colaboration) que procura identificar as classes do sistema, a sua interface e as relações entre elas. As principais atividades consistem em avaliar a especificação do cliente; extrair classes candidatas por análise da especificação; identificar grupos de classes (super classes); definir e atribuir responsabilidades para cada classe; identificar relações entre classes; identificar colaborações entre classes com base nas responsabilidades; e construir representações hierárquicas das classes.

Para além destas metodologias, sem dúvida alguma que aquela que atualmente é mais relevante é a proposta pela Rational (e que na realidade se baseia em muitos dos conceitos aqui referidos). Outras que vale a pena mencionar

são a de Shlaer e Mellor (1988), a de Edward Berard (1993) e a de James Martin (1992).

2.1 Metodologia RUP

Durante a década de 90, tornou-se óbvia para muitos teóricos da engenharia de software a vantagem do paradigma da orientação por objetos e, nesse sentido, começaram a proliferar as metodologias que tinham por base os respectivos conceitos. O exagerado número de metodologias que, entretanto foram propostas fez recordar a experiência negativa que já tinha ocorrido com as metodologias estruturadas, pelo que ganhou força a idéia da criação de uma metodologia de desenvolvimento de software “unificada”, que tirasse partido da experiência e dos conceitos da linguagem UML.

O RUP é uma metodologia de engenharia de software desenvolvida e comercializada pela empresa Rational Software. Tendo em conta que a construção de software de qualidade de uma forma repetitiva e previsível é difícil, e que as causas dos problemas associados a este tipo de desenvolvimento têm sido uma constante ao longo do tempo, o RUP propõe várias boas práticas para o desenvolvimento de software e aplicadas de forma integrada.

O RUP é mais do que uma "simples" metodologia de desenvolvimento de software, uma vez que pode funcionar como um conjunto de princípios genéricos utilizados para instanciar e configurar várias metodologias concretas, conforme o tipo de organização, o domínio de aplicação, o nível de competências, etc.

As Características Principais - O RUP suporta diversas boas práticas do desenvolvimento de software: (1) é uma metodologia de desenvolvimento de software iterativa; (2) propõe a gestão integrada de requisitos desde a sua identificação até à implementação; (3) propõe o desenvolvimento de software baseado em arquiteturas de software e em componentes; (4) defende a modelação visual; e (5) o controle de qualidade permanente. Além destas características, o RUP integra outras idéias fundamentais, nomeadamente o fato de ser orientado por casos de utilização.

As 4+1 Visões do RUP - Existem diferentes perspectivas segundo as quais o processo de desenvolvimento de software e o produto dele resultante podem ser encarados, sobretudo tendo em conta os interesses particulares dos diversos intervenientes no processo, que resultam das suas funções na organização. A definição de uma arquitetura sólida é a chave de sucesso para qualquer projeto que utilize uma abordagem orientada por objetos, segundo Booch (1995). O RUP apresenta um modelo segundo o qual considera a existência de 4+1 visões do sistema, definidas ao longo do processo, e cuja representação é seguinte: Visão Lógica, Visão Processo, Visão Implementação, Visão Distribuição e por fim a Visão Casos de Utilização. Cada visão é concretizada através de diversos diagramas do UML.

Para os usuários o mais importante é a satisfação dos seus requisitos funcionais, expressos numa visão lógica (Logical View) do sistema. A este nível é necessário garantir que se tem informação sobre o que o sistema deve fazer. A arquitetura lógica do sistema é representada através dos diagramas de classes que modelam as principais abstrações do sistema (classes, relações e pacotes).

Para os técnicos que participam no desenvolvimento do sistema, interessa ter a visão da organização do mesmo em termos dos módulos de software (desde o código fonte até aos executáveis), bem como resolver questões relacionadas com a gestão da configuração, a facilidade do desenvolvimento, a reutilização e restrições relacionadas com as linguagens de programação. Esta é a visão de implementação (Implementation View), que reflete a estrutura estática do sistema. Os principais elementos de modelação nesta visão incluem os pacotes e os diagramas de componentes.

A visão de processamento (Process View) do sistema preocupa-se em representar os conceitos relacionados com a implementação do sistema no seu ambiente de produção, como tarefas, atividades e processos, bem como com as suas interações. Nesta visão, são consideradas questões como paralelismo e concorrência na execução, tolerância a falhas, distribuição de objetos, tempos de resposta e escalabilidade, desempenho, confiabilidade e integridade do sistema. Esta visão é modelada por diagramas de componentes.

A visão de instalação (Deployment View) representa a correspondência entre os componentes desenvolvidos pelo projeto e o respectivo suporte tecnológico, modelando a configuração dos elementos de processamento em tempo de execução. As principais preocupações são questões como a migração e instalação do sistema e o respectivo desempenho, disponibilidade, confiabilidade e escalabilidade. São utilizados diagramas de componentes e de instalação, que permitem visualizar a localização física de componentes na infra-estrutura física e/ou computacional da organização.

Finalmente, a visão dos casos de utilização (Use Case View) tem em consideração os casos de utilização importantes, de forma a definir, por um lado, a arquitetura nas fases iniciais do processo e, por outro, a validar mais tarde a percepção das restantes visões. Funciona assim como a visão integradora de todas as restantes, e reforça a idéia do RUP enquanto metodologia conduzida por casos de utilização, que são naturalmente os principais diagramas de modelação desta visão.

2.2 Metodologia Iconix

O ICONIX é uma metodologia de desenvolvimento de software promovido pela empresa ICONIX Software Engineering, cujo foco de negócio reside na formação e produção de material para educação em tecnologias de objetos, em particular em CORBA, COM, Java e UML. O principal evangelista do ICONIX é Doug Rosenberg (1999), que define o ICONIX como um “processo” de desenvolvimento de software prático, algures entre a complexidade e abrangência do RUP (Rational Unified Process) e a simplicidade e o pragmatismo do XP (Extreme Programming).

O ICONIX é conduzido por casos de utilização, iterativo e incremental, tal como o RUP, mas sem a complexidade que este preconiza. Por outro lado, é relativamente pequeno e simples, tal como o XP, mas sem eliminar as tarefas de análise e de desenho que aquele não contempla. Por fim, o ICONIX usa o UML como linguagem de modelação e apresenta um alto grau de rastreabilidade (traceability).

Visão Geral - O ICONIX é uma metodologia de desenvolvimento de software, englobando as seguintes tarefas principais: (1) análise de requisitos; (2) análise e desenho preliminar; (3) desenho; e (4) implementação, conforme se introduz sumariamente em seguida.

A metodologia consiste na produção de um conjunto de artefatos que retratam as visões dinâmica e estática de um sistema, e que vão sendo desenvolvidos incrementalmente e em paralelo, os modelos da estática e os modelos da dinâmica.

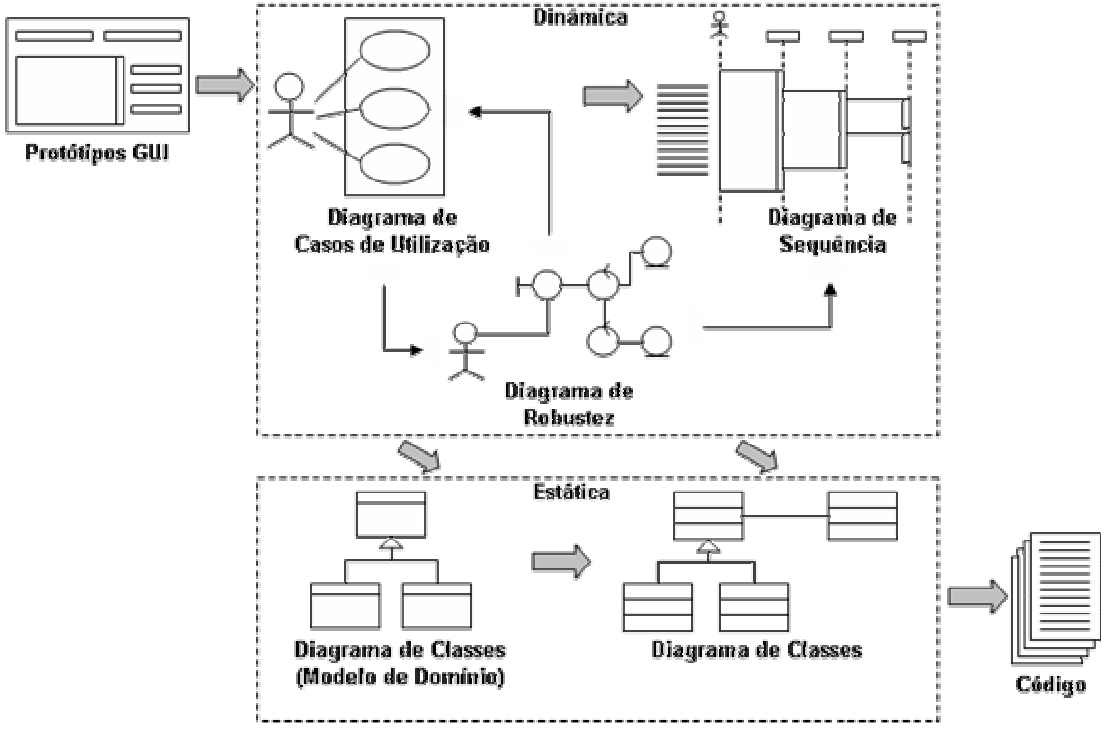

Figura 2.1: Visão Geral do Iconix (Silva, 2001).

A Figura 2.1 ilustra a visão geral do ICONIX. Esta figura revela um aspecto importante da utilização do UML: o fato da implementação de um sistema apenas depender da versão detalhada do diagrama de classes final. (Parece óbvio, mas muitos indivíduos pensam ainda que se poderiam usar, por exemplo, diagramas de seqüência para gerar código automaticamente!).

Análise de Requisitos - A tarefa de análise de requisitos consiste na realização das seguintes atividades (ver Figura 2.2):

- Identificar os objetos do mundo real e todas as relações de generalização, associação e agregação entre esses objetos. Desenhar o correspondente diagrama de classes de alto nível, designado por modelo de domínio.

- Se for razoável (e.g., se for pertinente ou se houver orçamento para tais atividades), desenvolver protótipos de interface homem-máquina (GUI), diagramas de navegação, etc., de forma que os utilizadores e clientes possam entender melhor o sistema pretendido.

- Identificar os casos de utilização envolvidos no sistema. Desenhar os diagramas de casos de utilização realçando os atores envolvidos e as suas relações.

- Organizar em grupos os casos de utilização. Capturar essa organização através de diagramas de pacotes (packages).

- Associar requisitos funcionais aos casos de utilização e aos objetos do domínio.

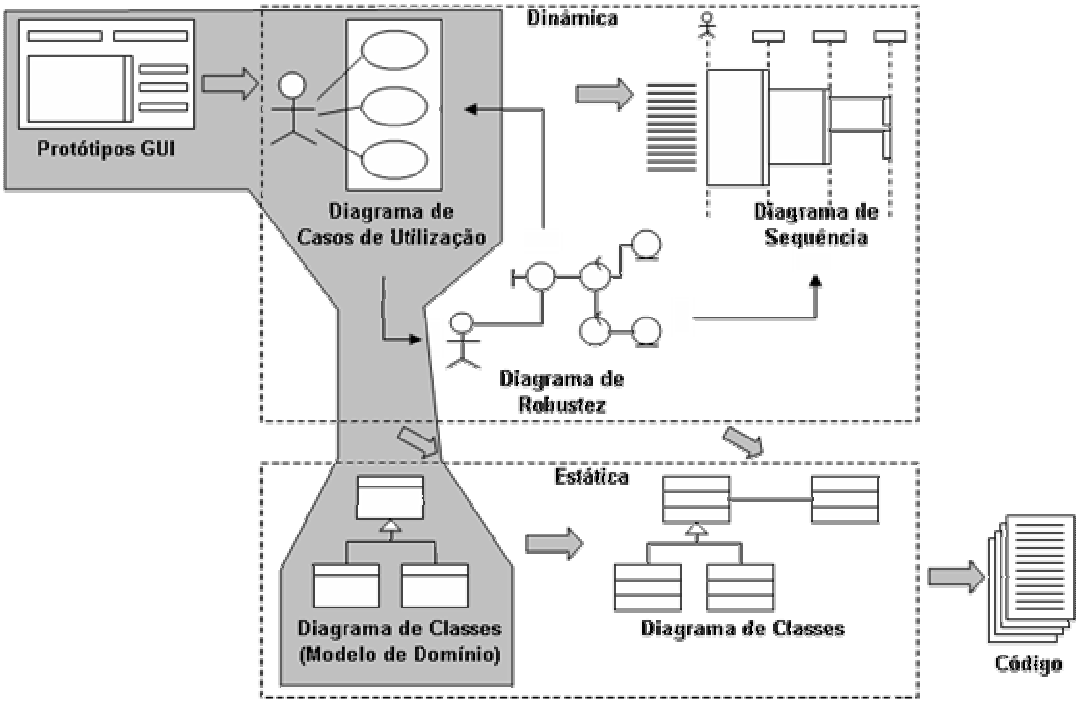

Figura 2.2: Atividades da tarefa de análise de requisitos (Silva, 2001).

Observação: O ICONIX distingue explicitamente um requisito de um caso de utilização. Em particular, o seu autor distingue-os da seguinte forma:

- Um caso de utilização descreve uma unidade de comportamento.
- Um requisito descreve uma regra que governa o comportamento.
- Um caso de utilização satisfaz um ou mais requisitos funcionais.
- Um requisito funcional pode ser satisfeito por um ou mais casos de utilização.

Ou seja, segundo o ICONIX, há uma relação de muitos-para-muitos entre casos de utilização e requisitos, pelo que tem sentido a última atividade desta tarefa, de associação entre estes dois conceitos. Note-se que existem outros autores que consideram, por outro lado, os casos de utilização o mecanismo ideal para se especificarem os próprios requisitos de um sistema. (Esta relação entre casos de utilização e requisitos é um assunto em discussão na comunidade OO, não existindo até ao momento qualquer consenso.).

Análise e Desenho Preliminar - A tarefa de análise e desenho preliminar consiste na realização das seguintes atividades (ver Figura 2.3):

- Fazer as descrições dos casos de utilização com os cenários principais, cenários alternativos e cenários de exceções.
- Fazer a análise de robustez. Para cada caso de utilização:
    - Identificar um primeiro conjunto de objetos. Usar os estereótipos de classes definidos no perfil “Processos de Desenvolvimento de Software” especificado no UML 1.3 (boundary, control, e entity).
    - Atualizar o diagrama de classes do modelo do domínio, com os novos objetos e atributos, entretanto descobertos.
- Terminar a atualização do diagrama de classes de modo a refletir a conclusão da fase de análise.

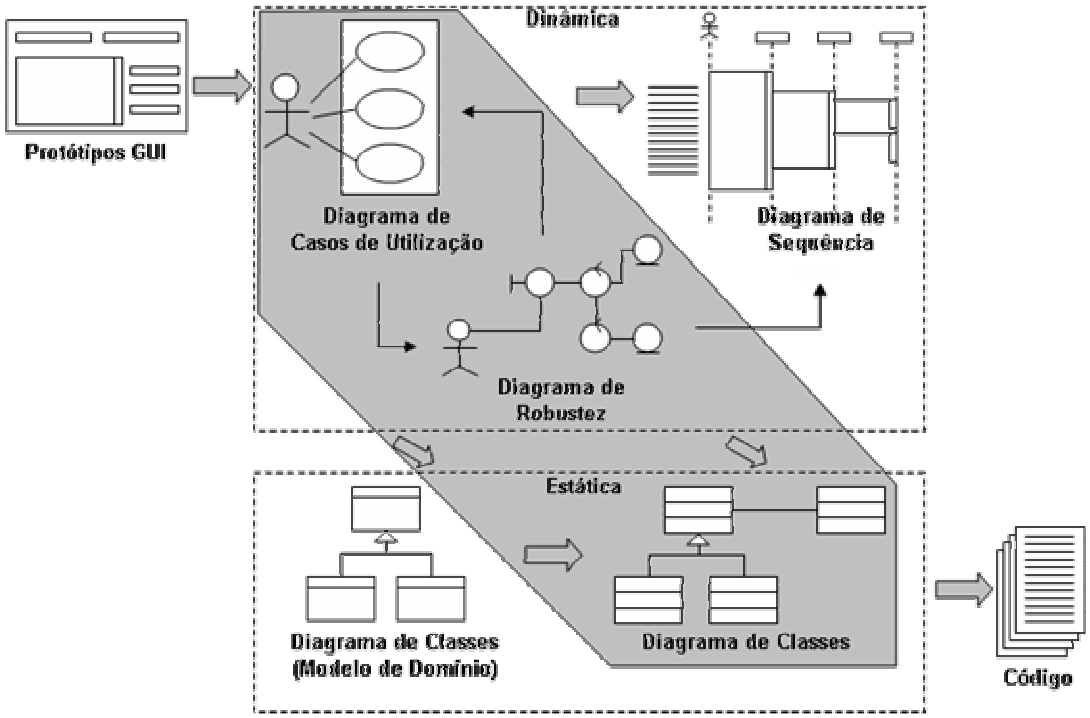

Figura 2.3: Atividades da tarefa de análise e desenho preliminar (Silva, 2001).

Desenho - A tarefa de desenho consiste na realização das seguintes atividades (ver Figura 2.4):

- Especificar o comportamento. Para cada caso de utilização:
    - Identificar os objetos, as mensagens trocadas entre objetos e os métodos associados que são invocados. Desenhar um diagrama de seqüência com o texto do caso de utilização do lado esquerdo, e informação do desenho do lado direito. Continuar a atualizar o diagrama de classes com os objetos e atributos, entretanto descobertos.
    - Se for relevante, usar diagrama de colaboração para ilustrar as transações principais entre objetos.
- Terminar o modelo estático, adicionando informação detalhada sobre o desenho (e.g., visibilidade e padrões de desenho).

- Verificar que o desenho satisfaz todos os requisitos identificados.

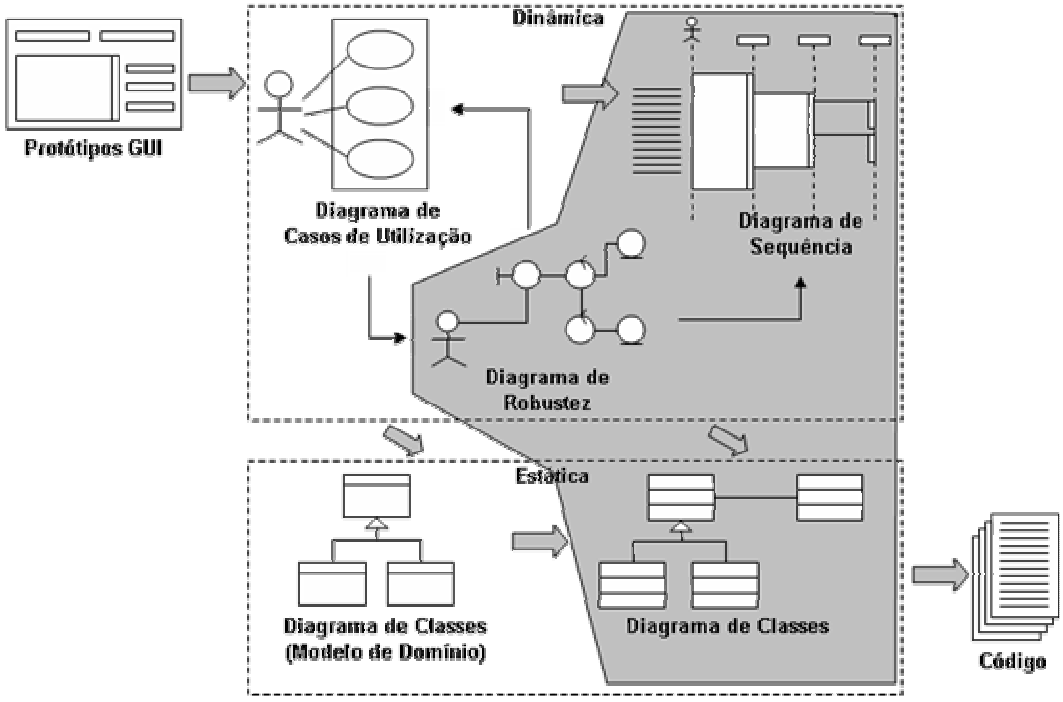

Figura 2.4: Atividades da tarefa de desenho (Silva, 2001).

Implementação - A tarefa de implementação consiste na realização das seguintes atividades (ver Figura 2.5):

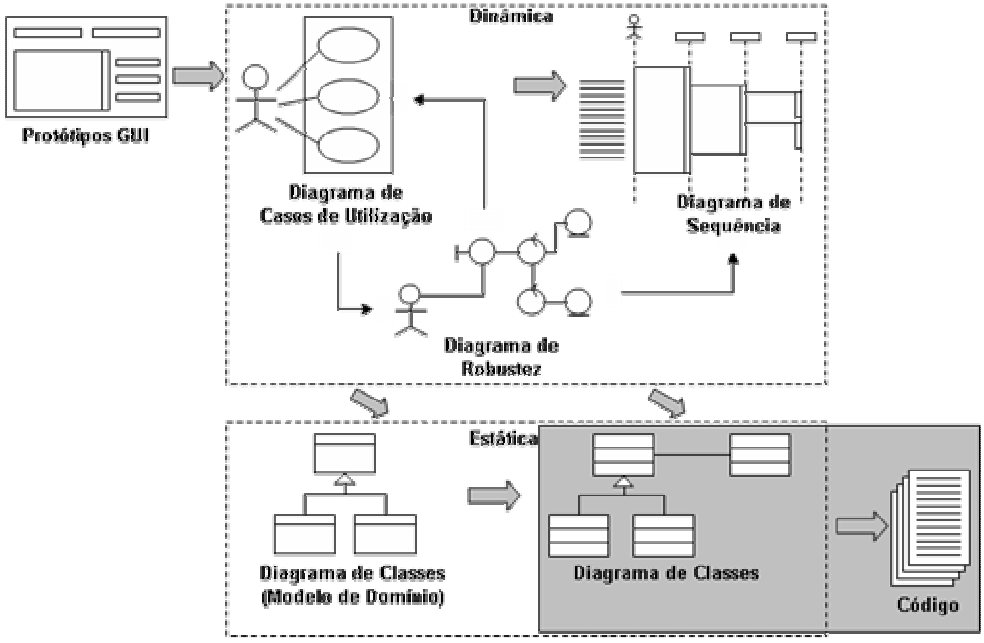

Figura 2.5: Atividades da tarefa de implementação (Silva, 2001).

- Consoante as necessidades, produzir diagramas de arquitetura, tais como, diagramas de instalação e de componentes, que apóiem a tarefa de implementação.
- Escrever e, eventualmente, gerar o código.
- Realizar testes unitários e de integração.
- Realizar testes de sistema e de aceitação do utilizador.

Avisos do Processo ICONIX - O ICONIX lança um conjunto de avisos relativamente a problemas e dúvidas comuns, que deverão ser tidos em conta pelos intervenientes do processo. Os avisos relacionados com a realização da tarefa de análise de requisitos têm como principal objetivo evitar a perda de tempo com detalhes desnecessários. Designadamente:

- Na produção dos modelos de domínio:
    - Não perder demasiado tempo com a inspeção gramatical.
    - Não endereçar o desenho da multiplicidade, demasiado cedo no projeto.
    - Endereçar a agregação e composição apenas na fase do desenho detalhado.
- Na produção dos modelos de casos de utilização:
    - Não começar a escrever os casos de utilização até se conhecer bem como é que os utilizadores irão atuar.
    - Não passar semanas a construir modelos de casos de utilização elaborados e bem desenhados, mas a partir dos quais não é possível construir-se um adequado desenho de classes.
    - Não perder muito tempo em discussões sobre quando e onde usar relações “include” ou “extend”.

- Não usar templates textuais de casos de utilização muito longos ou complexos.

Avisos a ter em conta na realização da tarefa de análise e desenho preliminar:

- Não procurar fazer desenho detalhado nos diagramas de robustez
- Não perder tempo a tentar aperfeiçoar os diagramas de robustez à medida que o desenho evolui.

Avisos a ter em conta na realização da tarefa de desenho:

- Na produção dos diagramas de seqüência:
  - Não procurar associar comportamento aos objetos antes de se ter um idéia concreta sobre o seu significado e interesse para o sistema.
  - Não começar a desenhar um diagrama de seqüência antes de se ter completado o diagrama de robustez correspondente.
  - Não focar a atenção (e esforço) na definição de métodos “get” e “set” em detrimento dos métodos reais. (Estes métodos de acesso e alteração dos atributos podem ser facilmente gerados automaticamente a partir de uma ferramenta CASE.).
- Na produção dos diagramas de estado:
  - Não desenhar diagramas de estados para objetos com apenas dois estados.
  - Não modelar o que não é necessário modelar.
  - Não desenhar diagramas de estados só porque se consegue desenhá-los.

2.3 Metodologia Fusion

A Metodologia (ou Método) Fusion de orientação a objetos foi proposta por Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes e Paul Jeremaes no ano de 1994, embora seu "nascimento" deu-se nos laboratórios da HP em meados de 1993. O método compreende as fases de análise, projeto e implementação (Coleman, 1996).

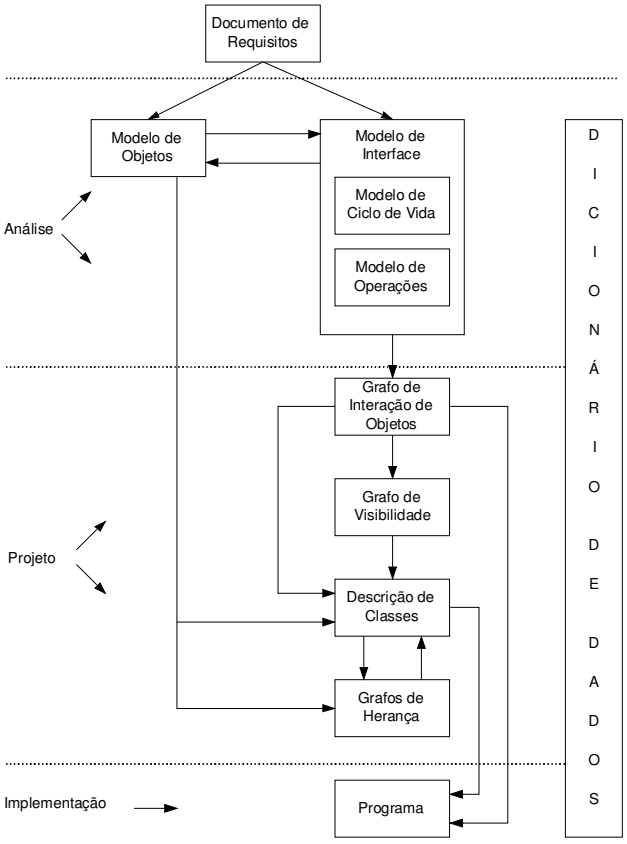

Figura 2.6 –Visão Geral do Método Fusion (Cotton, 1996).

O Fusion foi pensado para solucionar problemas encontrados nos métodos orientados a objeto de primeira geração, que são a "notação" e o "processo". Portanto, o Fusion fornece um processo para desenvolvimento de software bastante conciso, propondo notações bem definidas, completas e ferramentas para gerenciamento do processo que fazem a verificação e a consistência do mesmo.

O Fusion não encontra dificuldades em desenvolver sistemas seqüenciais. O mesmo não pode ser dito dos sistemas concorrentes, os quais nem todos podem ser bem representados. Neste caso, o próprio Derek Coleman sugere outras técnicas e métodos.

O Fusion recebeu influências dos seguintes métodos (Coleman, 1996):

- Object Modeling Technique (OMT): A fase de análise do Fusion foi inspirada no método OMT, porém com algumas mudanças. As influências podem ser notadas no modelo de objetos, no modelo de operações (análogo ao modelo funcional) e nos processos.

- Booch: Este método também influenciou a fase de análise do Fusion. A maior influência pode ser notada quanto à visibilidade dos objetos (Grafos de Visibilidade).

- Class Responsability Collaboration (CRC): O processo sistemático adotado pelo Fusion para a fase de projeto é baseado nesta técnica. Outro ponto é a influência nas interações entre os objetos.

- Métodos Formais: No modelo de operações, o qual será exposto na Seção 3.1.3.2, é que se nota a maior contribuição destes métodos com relação à criação do Método Fusion. Mais precisamente, estas contribuições encontram-se nas pré e pós-condições (cláusulas assumes e result, respectivamente) do modelo de operações.

Estas influências estão representadas graficamente na Figura 2.2.

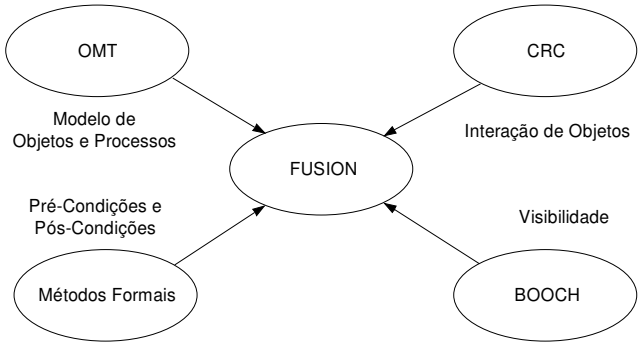

Figura 2.7 – Influências sobre o Fusion (Coleman, 1996).

2.3.1 Fase de Análise

A função da fase de análise é, baseada no documento de requisitos, identificar as classes e os objetos existentes no sistema, descrever os relacionamentos e definir as operações que poderão ser executadas pelo sistema. A fase de análise mostra o “que” o sistema faz e não “como” ele é feito. Obtém-se, no final desta fase, um “documento de especificações”.

A análise é uma fase que deve ser executada de forma iterativa e incremental. Durante todo o processo da análise ela deve ser acompanhada pela construção do dicionário de dados, que tem a função de identificar e descrever as entidades existentes.

O desenvolvimento incremental consiste em (Albuquerque, 1999):

- Especificações não completas do sistema no início do processo;
- Domínio do sistema crescente com o passar do tempo;
- Análise e projeto realizados interativamente;
- Modelos são detalhados a cada etapa, sendo baseados em anteriores;
- Alterações ou correções realizadas no decorrer do processo;
- Uso de protótipos.

O Método Fusion parte do princípio de que já exista um documento de requisitos pronto. “Vale lembrar que a captura de requisitos será normalmente executada por um usuário, o qual deve fornecer o documento inicial de requisitos; portanto, o Fusion não possui uma fase de captura de requisitos” (Coleman, 1996).

A importância de um bom levantamento dos dados pode ser exemplificada através de uma frase do físico Albert Einstein, na obra A Evolução da Física de 1938: “A formulação de um problema é muitas vezes mais essencial do que sua solução, a qual pode ser meramente uma questão de habilidade matemática ou experimental”.

Dois **modelos** são gerados pela fase de análise, os quais tratam de aspectos diferentes: o Modelo de Objetos – define a estrutura estática das informações do sistema – e o Modelo de Interfaces – define a comunicação (comportamento) das entradas e saídas do sistema.

Esta fase do processo é responsável pela descrição de:

- Classes de objetos, sem associar quaisquer métodos às classes;
- Relacionamentos entre classes;
- Operações que podem ser executadas;
- Seqüência de operações permitidas no sistema.

Os modelos da fase de análise se inter-relacionam (Figura 2.8) através de algumas características, sendo a verificação destes resumida nos seguintes tópicos (Coleman, 1996):

- Inteireza em relação aos requisitos:
    - Todos os cenários estão no ciclo-de-vida;
    - Todas as operações do sistema foram definidas;
    - Todas as informações estáticas estão compreendidas no modelo de objetos do sistema;
    - Todas as outras informações se encontram no dicionário de dados.
- Consistência simples:
    - Todos os conceitos (classes, relacionamentos, atributos, predicados) estão definidos no dicionário de dados ou no modelo de objetos do sistema;
    - A fronteira do modelo de objetos do sistema é consistente com o modelo de ciclo-de-vida;

- Todos os identificadores encontram-se no dicionário de dados.

- Consistência semântica:

  - Os eventos gerados pelo modelo ciclo-de-vida e pelo modelo de operações devem estar consistentes;

  - Modelo de operações deve preservar as definições do modelo de objetos do sistema;

  - Conferir os cenários manualmente através dos esquemas.

Vale lembrar que as verificações citadas acima devem ser observadas durante todo o processe de análise.

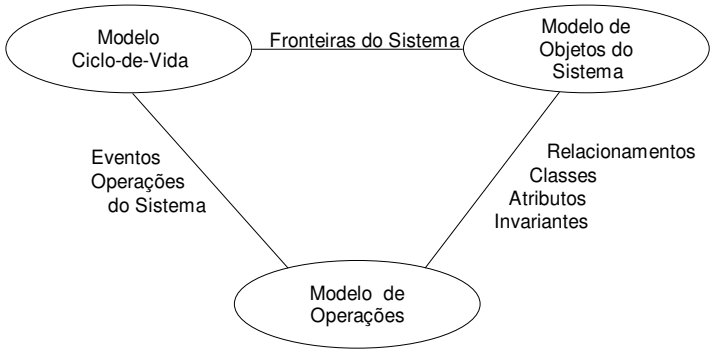

Figura 2.8 – Verificação de intersecções entre modelos (Coleman, 1996).

Dicionário de Dados - O sistema, ao propor um dicionário de dados, deve garantir a consistência dos dados que armazena, coibindo ações consideradas ilegais. O dicionário de dados serve como ferramenta de auxílio para os “leigos” em relação ao sistema em desenvolvimento.

A estrutura básica de um dicionário de dados (Figura 2.9) compreende: “nome”, “categoria” e “descrição” do termo ou conceito nele referenciado. Além destas, outras colunas podem ser introduzidas, com a finalidade de atender a requisitos impostos por algum tipo específico de item.

Nome	Categoria	Descrição
Nome do item	Classe, agente de operação do sistema, etc.	Texto definido ou explicando o item

Figura 2.9 – Estrutura de um Dicionário de Dados (Coleman, 1996).

Modelos de Objetos - Os conceitos representados neste modelo são: classes, relacionamentos, atributos, agregação, generalização/especialização e restrições de cardinalidade.

Modelo de Objetos do Sistema - Este modelo origina-se do modelo de objetos, já previamente construído. Ele modela apenas o subconjunto de objetos relacionados com o sistema, excluindo todas as classes e relacionamentos que pertencem ao ambiente do sistema. A notação utilizada para delimitar os limites do sistema e do ambiente do mesmo é a utilização de uma linha tracejada fechada na fronteira entre o sistema e o ambiente. Se o modelo de objetos do sistema venha a englobar a totalidade do modelo de objetos, significa que o sistema não deverá interagir com o seu ambiente, sendo, portanto, uma simulação.

Modelo de Interfaces - O modelo de interfaces descreve o comportamento do sistema, representando os eventos e as mudanças de estado por eles causados. Um evento, quando recebido pelo sistema – chamado de “evento de entrada” –, pode causar uma mudança de estado (“operação do sistema”) e quando o evento é gerado pelo sistema, é chamado de “evento de saída”. O modelo de interfaces utiliza-se de cenários e linhas de tempo que modelam os eventos entre os agentes e o sistema.

Dois modelos são utilizados, como parte do modelo de interfaces, para modelar o comportamento do sistema: modelo de operações e o modelo de ciclo-de-vida. O modelo de ciclo-de-vida descreve o comportamento do sistema com o ambiente desde a sua criação até o seu final, definindo as seqüências de interações permitidas. Se um evento não permitido vier a ocorrer, este será ignorado sem alterar o estado do sistema. O modelo de operações caracteriza-se pela

representação do comportamento das operações do sistema através de “mudanças de estado” e “eventos gerados” (Coleman, 1996).

2.3.2 Fase de Projeto

Decide como representar as operações do sistema através de interpretações entre objetos relacionados e como estes obterão acesso entre si. Como resultado desta fase, obtém-se um “documento de arquitetura”. O resultado da fase de projeto também pode ser interpretado como a estrutura de software orientada a objetos contendo as mesmas informações e preservando os relacionamentos do modelo de objetos do sistema. O projeto ainda define como a funcionalidade definida no modelo de operações será implementada. A fase de projeto utiliza-se de 4 modelos:

- Grafos de Interação de Objetos - têm a finalidade de definir a seqüência de mensagens que são trocadas entre os objetos de um determinado grupo, quando da realização de uma determinada operação. Para o desenvolvimento de um software mais confiável, deve-se ter um GIO para cada operação do sistema, pois estes fornecem uma representação visual para a futura implementação;

- Grafos de Visibilidade - O objetivo de um Grafo de Visibilidade (GV) é definir as estruturas de referência entre as classes existentes no sistema. Um objeto pode ser tanto cliente como servidor. Todo objeto servidor deve estar visível a todo e qualquer objeto cliente, para que este seja capaz de enviar mensagens aos objetos servidores;

- Descrição de Classes - as informações do modelo de objetos, dos grafos de interação de objetos e dos grafos de visibilidade são consolidadas nas descrições de classes. Neste estágio são estabelecidos para cada classe: superclasse(s) imediata(s), seus métodos, seus atributos de dados e atributos-objeto. Após isto, serão atualizadas as informações relativas à herança;

- Grafos de Herança - O conceito de grafos de herança significa que uma classe pode ser definida como uma especialização de outra.

2.3.3 Fase de Implementação

A Fase de Implementação consiste em codificar o projeto em uma linguagem de programação. A base para isto é composta pela descrição de classes, grafos de interação de objetos e dicionário de dados, da fase de projeto e o ciclo-de-vida gerado na fase de análise.

Embora não haja uma ordem determinada a ser seguida na fase de implementação, é conveniente utilizar-se uma seqüência de eventos. Para ordenar os eventos, dando preferência primeiramente às operações mais simples, utiliza-se o ciclo-de-vida. A partir de então, implementa-se as classes que irão suportar estas operações. Após, implementa-se os outros eventos, o que poderá vir a forçar a modificação das classes já implementadas, até que se complete as definições preestabelecidas. Como resultado desta fase, tem-se o “código fonte” do sistema requerido.

Uma ferramenta CASE que suporta as notações do Fusion é a WinA&D.

2.4 XP – Xtreme Programming

Segundo Bona(2003) o Extreme Programming (XP) é uma metodologia leve, eficiente, flexível e de baixo risco para times pequenos e médios, que desenvolvem software com requisitos dinâmicos ou em constante mudança.

A metodologia XP foi criada por Kent Beck, que pensava no início dos anos 1990 sobre desenvolver software de uma maneira mais fácil. Em 1996, Kent começou um projeto usando novos conceitos em desenvolvimento de software, que resultaram na metodologia XP (WEELS, 2002).

De acordo com Beck (2000), XP se distingue de outras metodologias por:

- apresentar feedback (retornos) contínuos e concretos em ciclos curtos;
- abordar planejamento incremental, apresentando rapidamente um plano global, que evolui durante o ciclo de vida do projeto;
- ter habilidade flexível de programar implementação de funcionalidade, respondendo as mudanças das regras de negócio;
- confiar nos testes automatizados escritos pelos programadores e clientes, para monitorar o progresso do desenvolvimento, permitindo a evolução do sistema e detectando antecipadamente os problemas;
- acreditar na comunicação oral, na colaboração íntima dos programadores, nos testes e no código fonte, definindo a estrutura do sistema e os objetivos;
- confiar no processo de evolução do projeto, que dura tanto quanto o sistema;
- acreditar nas práticas que trabalham tanto com as aptidões, a curto prazo, dos programadores, quanto os interesses, a longo prazo, do projeto.

XP é uma disciplina de desenvolvimento de software baseado em valores de simplicidade, comunicação, feedback e coragem. XP envolve o time inteiro para um trabalho de equipe com práticas simples, com feedback suficiente para capacitar o time a ver onde eles estão e a convergir para práticas em uma solução única (JEFFRIES, 2001).

No XP existe o conceito de um time único onde cada contribuinte do projeto é um integrante do time inteiro. Existe a figura do “Cliente”, que participa ativamente com o time diariamente. O planejamento e acompanhamento são simples e ajudam a decidir qual a próxima tarefa a ser realizada e predizer quando o projeto será feito. Definidas as regras de negócio, o time produz o software em uma série de pequenas versões, que passam por todos os testes que o cliente definiu.

Ainda de acordo com Jeffries (2001), os programadores trabalham em pares e em grupo, com projeto simples e código obsessivamente testado, melhorando o projeto continuamente para manter sempre as necessidades correntes e o sistema integrado. Os programadores escrevem todo código de produção em pares, e todos trabalham junto o tempo todo. Eles codificam de forma consistente para que todos possam entender e melhorar o código quando necessário.

2.5 Outras Metodologias

Apesar dos benefícios que se reconhecem na utilização de metodologias (independentemente do paradigma utilizado), elas não estão isentas de críticas e de aspectos menos positivos:

- Complexidade nos conceitos, técnicas e aplicação.
- Desconhecimento global da metodologia e falta de competências dos informáticos para a sua execução com qualidade.
- Ferramentas que suportam a metodologia difíceis de utilizar.
- Constatação da ausência de melhorias significativas no processo e produto final.
- Concentração na análise da situação atual, menor importância aos objetivos futuros.
- Tempo que decorre até a disponibilização dos resultados finais.
- Rigidez na aplicação dos métodos e conceitos.

Como reação a esta situação, um novo grupo de metodologias começou a aparecer nos últimos anos, que implicam um nível de formalismo muito menor. Muitas delas advogam a não realização de atividades de análise e desenho, e a produção de muito menos documentação por comparação com as metodologias estruturadas ou orientadas por objetos. Defendem que a principal documentação de um sistema é, ou deveria ser, o código fonte das aplicações desenvolvidas.

Comungam entre si a idéia de que as principais atividades a realizar ao longo de todo o processo de desenvolvimento são essencialmente a programação e os testes.

Estes métodos são bastante adaptáveis, também como forma de responder adequadamente aquilo que eles consideram ser a imprevisibilidade dos requisitos ao longo do tempo. Concentram-se na satisfação das necessidades das pessoas (informáticos e utilizadores) e não na definição de processos. Partilham a idéia do desenvolvimento iterativo típico das abordagens orientadas por objetos, e reforçam a importância da atividade de testes.

Apesar delas partilharem um conjunto de idéias base comuns, não existe ainda uma designação formal para elas, a não ser o termo lightweight (leve ou simplificado). São na sua maioria desconhecidas do grande público informático, apesar de algumas delas estarem sendo aplicadas há vários anos. Algumas referências relevantes nesta área são as abordagens XP - Extreme Programming de Beck (1999), Feature Driven Development de Coad (1999) e DSDM - Dynamic System Development Method.

Tal como referido por Fowler (2000), em determinadas circunstâncias estes métodos são particularmente aconselháveis, sobretudo se estivermos falando de sistemas pequenos ou com requisitos incertos ou voláteis, em que os programadores são responsáveis, experientes e se encontram motivados e em que os clientes são igualmente participativos e responsáveis, ou ainda quando a equipe de desenvolvimento é relativamente reduzida e estável. Em outras situações será preferível um processo mais formal, nomeadamente sempre que o projeto exigir a alocação de equipes de maior dimensão, ou existir um contrato com âmbito e preços fixos, ou em que o risco seja elevado e o processo de controle do projeto deva ser reforçado.

2.6 Comparação de Metodologias

A comparação das diversas metodologias atualmente existentes é uma tarefa complexa devido a um conjunto de dificuldades que se colocam e das quais podemos destacar as seguintes:

- Não existem metodologias iguais e, portanto em qualquer comparação estaremos sempre a comparar conceitos por vezes não comparáveis.
- Muitas metodologias são influenciadas ou particularmente concebidas para serem utilizadas com linguagens de programação específicas.
- Muitas metodologias assumem um contexto de aplicação onde não existem os problemas que no mundo real têm que enfrentar.
- A abrangência das metodologias varia fortemente; algumas apenas descrevem um processo, outras incluem técnicas e notações.
- A comparação entre metodologias tem que considerar obrigatoriamente apenas um subconjunto das mesmas, e um subconjunto de funcionalidades.
- A própria definição do conceito metodologia pode ser limitativa.
- Hoje em dia reconhece-se a vantagem clara do ponto de vista conceitual das abordagens orientadas por objetos face às estruturadas, pois apresentam:
- Um único paradigma consistente ao longo de todo o processo, mais próximo do processo cognitivo humano.
- Facilitam a reutilização não apenas do código, mas também da arquitetura global do sistema, o que potencia o aumento de produtividade dos informáticos.
- Apresentam modelos que refletem mais adequadamente o mundo real.
- Não existe separação entre dados e processos, o conceito unificador agrega as duas visões.
- Os detalhes de implementação são escondidos do exterior pela aplicação de técnicas de encapsulamento da informação.

- A facilidade de realização das tarefas de manutenção é maior, em resultado das diversas características anteriormente enumeradas.

- O sistema construído é conseqüentemente mais estável.

Estas abordagens têm, contudo, os seus problemas, pois se reconhece que nem sempre é fácil encontrar os objetos e classes apropriados no domínio do problema, uma vez que a maioria dos informáticos continua a pensar em termos funcionais. Além disso, só recentemente começaram a surgir no mercado ferramentas de apoio ao processo de desenvolvimento segundo o paradigma da orientação por objetos.

2.7 Considerações Finais

A crescente complexidade dos sistemas de informação vem tornando cada vez mais relevante o papel das metodologias de desenvolvimento de sistemas de informação. A grande variedade de metodologias, associada à oferta de várias técnicas e ferramentas, leva à necessidade de um estudo comparativo destas metodologias.

Diferenciando-se pelo modelo referencial como também pelo enfoque utilizado na modelagem do sistema: orientado às funções, aos dados, aos objetivos, a objetos ou com outras orientações, a escolha da metodologia que mais se adapte às necessidades e recursos das empresas de desenvolvimento de software, torna-se extremamente difícil.

Uma boa sugestão é o método Fusion, criado por Coleman e outros com o objetivo de abranger o processo de desenvolvimento de software nas fases de análise, projeto e implementação. O conjunto de modelos sugeridos pelo método é conciso e a passagem de uma fase para outra é bem definida.

3 METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE SIMPLIFICADA

O desenvolvimento de software é um processo complexo. Os usuários precisam primeiramente decidir sobre os seus requisitos. A seguir devem ser especificados a equipe do projeto, o programa, o teste e implementação desses requisitos, ora mudando um sistema de computador existente ora criando um completamente novo.

É necessário planejar o projeto no começo e monitorá-lo durante sua vida para assegurar que os prazos sejam cumpridos e que o sistema final atenda às necessidades do negócio. Também, o sistema e sua documentação, devem aceitar mudanças e melhorias.

Os objetivos deste capítulo são:

- Apresentar esquemas, figuras de modelo de fábrica. Tecer comentários sobre cada um dos componentes, justificando a sua necessidade e mostrando a sua funcionalidade.

- Apresentar metodologia de desenvolvimento de software adequada ao modelo de fábrica e de fácil implementação, descrevendo as várias etapas do ciclo e caracterizando cada uma com as suas atividades.

Esta metodologia utiliza a notação UML e se baseou na metodologia FUSION. Ele possui um conjunto de modelos sugeridos conciso e a passagem de uma fase para outra é bem definida sendo sempre seguida de sugestões de atividades de inspeção para checagem da consistência dos modelos produzidos. Mas ele possui algumas desvantagens como a não utilização de uma notação padronizada, dificultando o entendimento e intercâmbio de seus modelos por parte das equipes de desenvolvimento. Também a falta de definições específicas para a extração dos requisitos do sistema contribui para que implantação do método seja dificultada. Por fim, a falta de uma fase de testes que permita validar o resultado da fase de implementação, é de suma importância para o processo de homologação do software.

Aqui o Fusion está sendo simplificado por que, freqüentemente os sistemas produzidos por empresas de pequeno e médio porte não permitem utilização do processo Fusion como um todo, devido a problemas como prazo de entrega. Na fase de análise as alterações consistem na implementação de atividades de extração de requisitos, utilização de diagramas de casos de uso, diagramas de seqüência e simplificação do modelo de operações, enquanto o modelo de ciclo-de-vida será descartado.

Estas modificações não significam que as atividades que foram deixadas de lado não têm valor significativo no processo de desenvolvimento de um software. Muito pelo contrário, o que se propõe com esta alternativa é simplificar certos passos na fase de análise.

Como resultado desta atitude, o sistema tem grandes chances de ser entregue no limite determinado, e ainda possuir um material significativamente importante para a sua posterior manutenção.

A seguir procurou-se apresentar o processo e descrever as atividades envolvendo a especificação de requisitos e análise dos módulos de aplicações.

3.1 Processo de desenvolvimento de software

Segundo Hans-Erik Eriksson (2000), o processo de desenvolvimento de software especifica as atividades a serem feitas no desenvolvimento do software, as instruções de como fazer estas atividades, os resultados destas atividades e a ordem de sincronização entre elas.

Um processo faz uso de uma notação. Processos mais avançados incluem também, *checklists*, *guidelines, metrics*, padrões de documentação e ferramentas recomendadas.

O processo de desenvolvimento proposto a seguir divide as atividades de desenvolvimento de software em 5 fases:

Especificação e análise – Onde são capturados os requisitos do sistema, vindos da arquitetura e do processo de negócio, e feita à criação de um modelo da solução encontrada, utilizando protótipos e diagramas para representá-la.

Projeto – Onde é definida a estrutura básica do sistema e feita a expansão do modelo vindo da análise, incluindo as soluções técnicas.

Implementação – Onde os modelos construídos são programados segundo a linguagem de programação escolhida. Inclui-se aqui, a atividade de revisão de código e teste unitário.

Testes – Aqui são executados os testes de integração e testes de sistema. O primeiro com o intuito de testar os subsistemas ou partes de um sistema que foi construído por diferentes programadores. O segundo serve para validar os requisitos de análise, atestando que o sistema foi construído corretamente.

Instalação – Que envolve as atividades que preparam o software para ser usado pelos usuários finais, como: empacotamento, instalação, documentação, treinamento e suporte ao usuário.

A figura 3.1 demonstra a seqüência do processo e a seguir será feito o detalhamento da fase de Especificação e Análise.

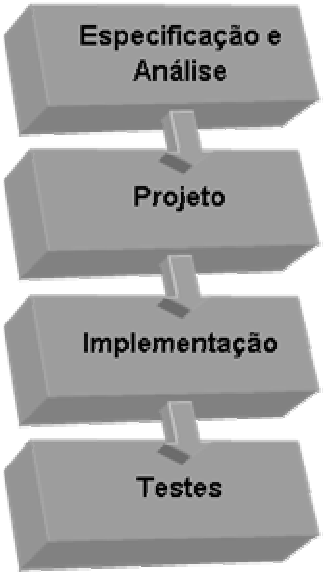

Figura 3.1: Seqüência do processo de desenvolvimento.

3.2 Especificação e Análise

As atividades de especificação e análise são o ponto de partida para o processo de desenvolvimento em si. A especificação e análise dão seguimento ao planejamento e devem ser sua primeira atividade.

Sua principal função consiste em descrever o que o sistema é e a sua funcionalidade. Sobre o que o sistema é, entende-se a estrutura propriamente dita, sendo representado de forma estática. Sobre a sua funcionalidade, entende-se as reações do sistema aos estímulos externos (eventos ou operações do sistema).

Esta é uma atividade de comunicação. Isto posto, cabe ao analista comunicar, através de uma documentação adequada, as características do sistema, obtidas através do seu entendimento do negócio, ao time de desenvolvimento, os projetistas. Resumindo, cabe a ele, informar o que é o sistema e o que deve resultar quando ele sofre eventos externos.

O analista não deve informar como o sistema atua para produzir os resultados. Esta função cabe ao projetista, bem como as preocupações arquitetônicas.

O escopo desta fase consiste em:

1. Informar os requisitos funcionais do sistema, de tal forma que possam ser rastreados por todo o desenvolvimento e confirmada a sua consecução.

2. Informar os requisitos não funcionais do sistema, de tal forma que possam ser rastreados por todo o desenvolvimento e confirmada a sua consecução – estes requisitos apontam para os parâmetros de qualidade do sistema.

3. Informar a estrutura estática do sistema, descrevendo o que ele consiste em ser.

4. Informar o comportamento dinâmico do sistema, através das respostas dadas aos diversos estímulos ocorridos.

5. Definir interfaces e, no caso de sistemas Web, navegação e mensagens com servidor.

Também cabe ao analista, a geração dos planos de testes, sendo que estes são tratados de forma específica mais adiante devido à sua magnitude.

Na figura 3.2 podemos observar um fluxo contendo todas as atividades a serem realizadas nesta fase de especificação e análise.

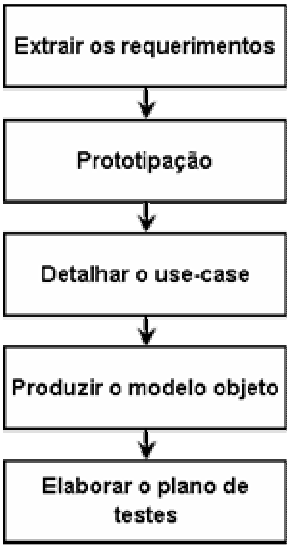

Figura 3.2: Fluxo da fase de especificação e análise

3.2.1 Extrair os requisitos

O primeiro passo do analista é encontrar os requisitos funcionais e não funcionais do sistema. Para que isto seja possível, o analista deve mapear o negócio. Esta é uma atividade que deve ser cumprida juntamente com o cliente, que é o principal fornecedor de como é o modelo de negócio. O analista também poderá valer-se de documentos, sistemas legados, plano de projeto ou qualquer outra fonte.

Sobre os requisitos funcionais, eles representam “o que” o sistema faz e é determinado, principalmente, pelas suas funcionalidades. E os requisitos não funcionais representam "do que é" e "do que faz", em relação a: precisão, desempenho, segurança, confiabilidade, manutenibilidade, portabilidade, robustez, resposta ao usuário, restrições, premissas, entre outros.

Dois documentos são produzidos nesta fase, o primeiro é chamado de descritivo funcional e abrigará todas as informações relacionadas ao mapeamento das características e funcionalidades do sistema; o segundo é chamado de Descritivo Não Funcional, que como o próprio nome diz, deverá conter uma descrição detalhada dos requisitos não funcionais do sistema.

Descritivo Funcional

A atividade de elaboração do descritivo funcional deve mostrar, em termos resumidos, as características do sistema. Isto é, descreve as necessidades de negócio e a interação com o usuário. Basicamente, deve incluir:

- Motivação do módulo ou sistema – porque este produto é bom para os negócios.
- Princípios do negócio em questão – estes princípios, que são derivados dos motivos, deverão originar os requisitos adiante.
- Objetivos do módulo – resumidamente, quais os objetivos a alcançar com o desenvolvimento deste produto.
- Comportamento – descrição do processo de negócio a ser atendido pelo sistema. Não é necessário o uso de diagrama de atividades, apenas uma visão resumida de, no máximo, uma ou duas páginas.

Para a produção dos artefatos desta atividade é necessário que se façam entrevistas com o usuário, documentando em uma ata, ou pode-se partir de uma ordem de serviço ou equivalente, desde que contenha as informações necessárias.

O analista tem a responsabilidade de preparar e convocar as reuniões ou entrevistas, analisar as entidades de negócio, assegurar que a informação gerada está de acordo com as necessidades do negócio, ajudar a definir a visão do escopo e estabelecer princípios e padrões.

O cliente tem o dever de fornecer o conhecimento do domínio (conceitos e termos de negócio, modelos, tendências, objetivos, métricas e princípios) e contribuir com a especificação do contexto de uso e necessidades a conceituar.

Os recursos necessários ao desenvolvimento desta atividade estão descritos no quadro 3.1.

Quadro 3.1 - Recursos necessários à produção do descritivo funcional.

Papéis	Ferramentas
Cliente	Metodologia JAD[2]
Analista	5WH[3]
	Editor de Textos
	Intranet de Documentação

É neste passo que se inicia formalmente a atividade de requisitos de análise, portanto, a equipe já deverá estar alocada.

O analista coordenador deverá convocar uma reunião com os seguintes participantes: Analista(s), Cliente responsável pelo negócio, Analista do cliente com domínio de negócio, e para módulos maiores, o Arquiteto de Soluções. Também é necessário um documentador.

A reunião deverá ser realizada no formato JAD, onde deverão ser questionadas: a visão de negócio, a motivação (o porquê do sistema a ser desenvolvido) e os princípios do negócio, que deverão ser estabelecidos e confirmados formalmente. A partir dos princípios estabelecidos serão derivados os objetivos do sistema e dos objetivos as métricas. Poderão ser utilizadas ainda, outras metodologias na busca da elicitação dos requisitos, como por exemplo, o método 5H1W. Enfim, deverá ser construído um fluxo do negócio, utilizando-se de um *storyboard*.

Deverão ser feitas quantas reuniões forem necessárias, a fim de obter um refinamento cada vez maior sobre o negócio. Ao final de cada reunião o descritivo funcional deverá ser confeccionado ou atualizado com novas informações.

---

[2] Joint Application Development – Conjunto de técnicas de dinâmica de grupo aplicadas em reuniões, visando agilizar o processo de elicitação de requisitos.

[3] Tipo de Check-list utilizado para garantir que a operação seja conduzida sem nenhuma dúvida por parte da chefias e subordinados. Os 5W correspondem às seguintes palavras do inglês: What (o que); Who (quem); Where (onde) When (quando) e finalmente Why (por que). O 1H corresponde a How (como), ou seja, método a ser utilizado para conduzir a operação.

Após uma revisão e aprovação pelo time, o analista deverá confeccionar um documento texto, denominado de Descritivo Funcional, sendo publicado na *intranet* de documentação. Na figura 3.3 podemos observar o fluxo desta atividade.

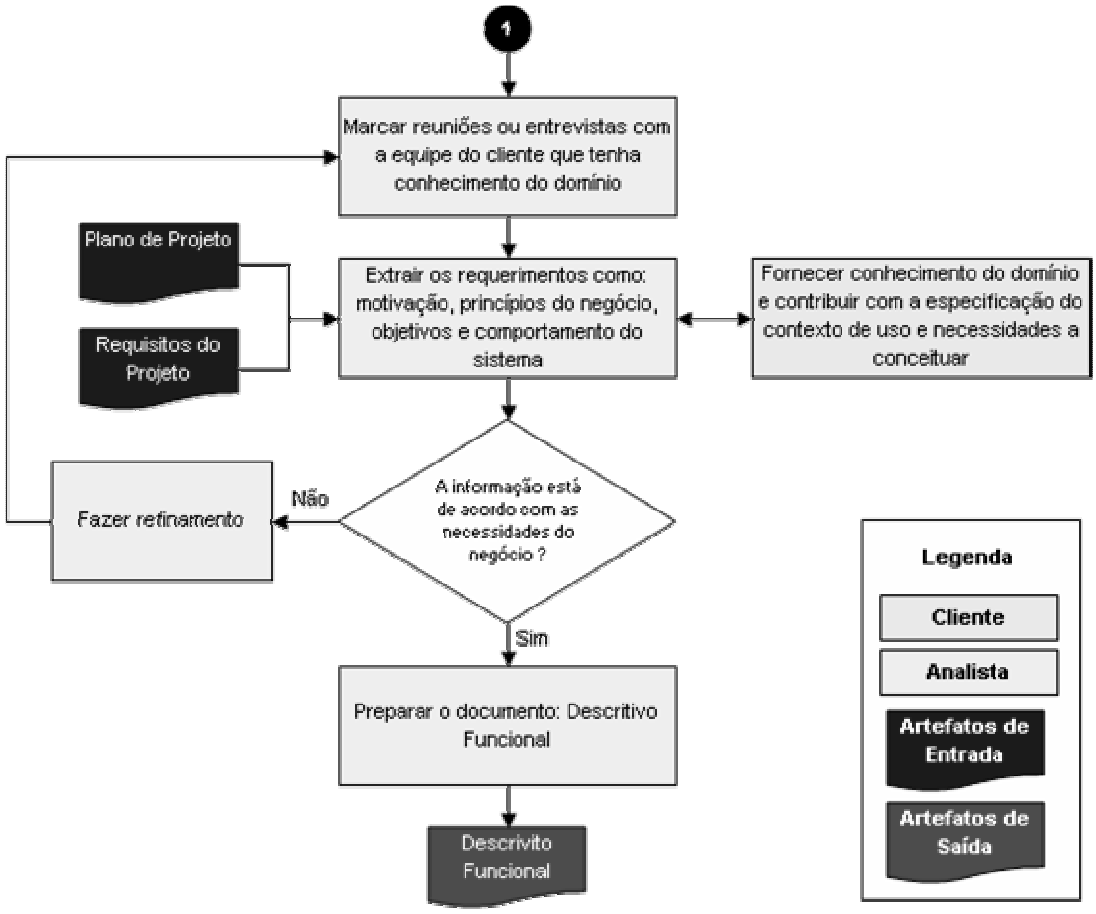

Figura 3.3: Fluxo da atividade de elaboração do descritivo funcional.

Descritivo não Funcional

A fase de extração dos requisitos é onde há a captura das necessidades do usuário, portanto, necessidades do sistema. Todos os demais produtos devem ser derivados destes requisitos adquiridos.

O descritivo não funcional deve conter uma lista de requisitos, conhecida por FLURPS, pois divide os requisitos em 7 categorias:

Funcionalidade (**F**unctionality) - Descreve as necessidades funcionais do sistema, o que ele necessita fazer para cumprir os objetivos do negócio. Por exemplo: *"O sistema deverá funcionar assim assado".*

Localizabilidade (**L**ocalization) - Descreve os objetivos referentes aos múltiplos ambientes e localizações, incluindo países. Por exemplo: *"O sistema deverá funcionar assim assado".*

Usabilidade (**U**sability) - Fatores como interface de usuário, facilidade de uso, documentação e material de ajuda. Por exemplo: *"O sistema deverá funcionar assim assado".*

Confiabilidade (**R**eliability) - Descreve o critério de liberação do produto que deve ser desenvolvido para qualquer objetivo de qualidade. Por exemplo: *"O sistema deverá funcionar assim assado".*

Performance (**P**erformance) - Critérios de performance são normalmente testados ao final do produto, entretanto, os critérios já devem ser pensados em fases iniciais do projeto. Por exemplo: *"O sistema deverá funcionar assim assado".*

Suportabilidade (**S**calability) - Quais os objetivos que garantirão um bom suporte ao sistema em tempo de execução. Quais os requisitos devem estar internos no sistema para que um help desk ou outras pessoas possam mantê-lo ou dar atendimento. Por exemplo: *"O sistema deverá funcionar assim e assado".*

Cada Requisito deve possuir as seguintes informações:

- Definição
- Escala - qual o parâmetro de medida do requisito.
- Teste/Ferramenta - como o requisito será medido.
- Produto Prévio – opcional - se houver um produto anterior, como era.
- Competição – opcional - se necessário, como a competição se comporta nesta área.

- Objetivo – dentro da escala qual o objetivo a ser seguido.
- Mínimo aceitável - dentro da escala qual o mínimo aceitável.
- Real – opcional - a ser preenchido no decorrer do projeto, o que foi realmente atingido.

Os artefatos de entrada desta fase são documentos, rascunhos de entrevistas ou reuniões com usuário e o Descritivo Funcional e os recursos necessários ao desenvolvimento desta atividade estão descritos no quadro 3.2.

Quadro 3.2 - Recursos necessários à produção da descrição dos requisitos.

Papéis	Ferramentas
Analista do Cliente conhecedor do Domínio Analista	Entrevistas JAD 5WH Descritivo Funcional Editor de Textos Intranet de Documentação

Para a execução desta atividade, é necessário que se marque uma ou mais reuniões com o usuário, no formato JAD, onde buscar-se-á identificar e classificar os requisitos não funcionais.

A cada requisito identificado, deve ser feito um detalhamento quanto à definição, qual o parâmetro de medida e como ele será medido, qual o mínimo aceitável e qual a medida meta, se já existe um produto anterior, como se comporta a competição e quando houver necessidade qual é realidade deste requisito.

Com todos os requisitos rastreados, o analista deverá transcrevê-los no Descritivo Não Funcional e apresentá-lo ao restante da equipe para avaliação. Deverão ser feitas quantas reuniões forem necessárias para refinar este documento até que esteja aprovado por toda a equipe. Após obter a aprovação, o analista deverá preparar um documento de aceite para que o cliente, na pessoa do gerente de projeto, dê o aceite tanto para o descritivo funcional quanto para o não funcional. Depois disto, resta ao analista publicar o documento na intranet de documentação.

Os produtos deste passo são os requisitos não funcionais identificados, detalhados, aceitos pelo usuário e publicados na intranet. Na figura 3.4 podemos contemplar as sub-atividades a serem realizadas.

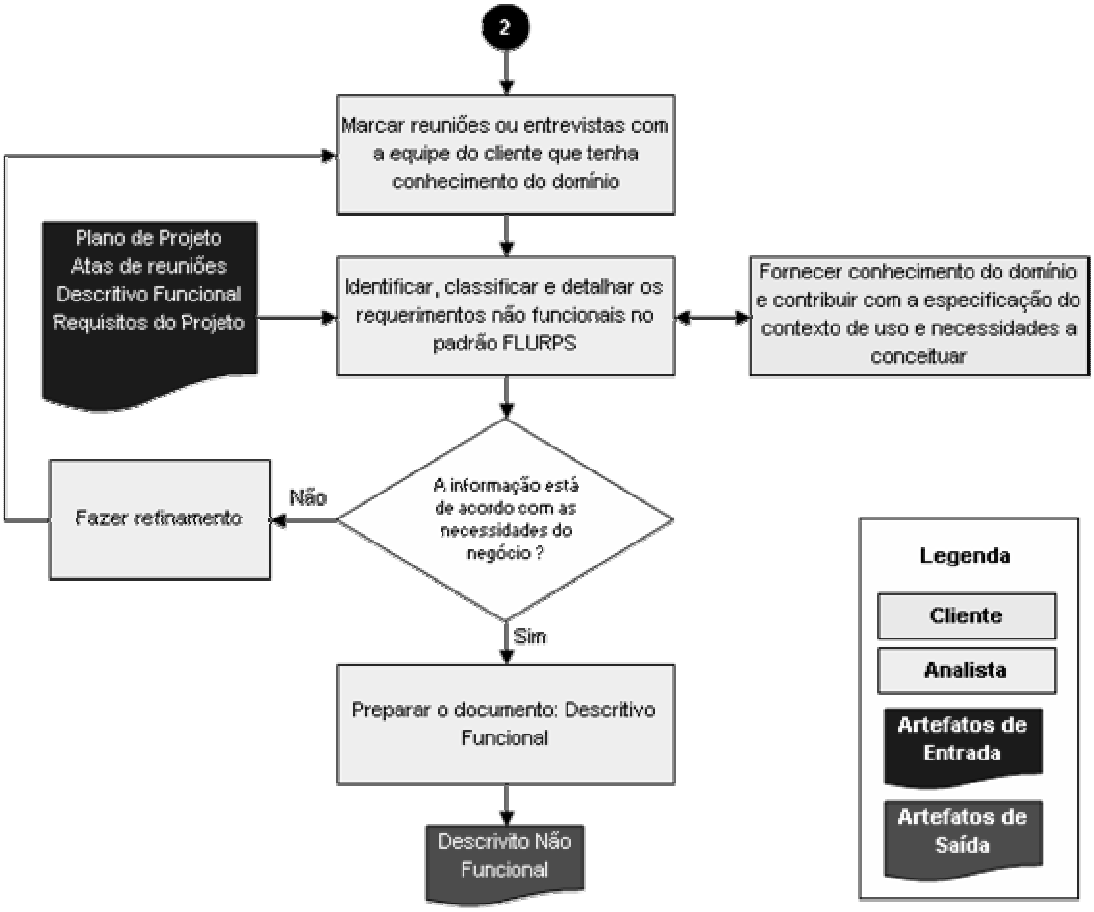

Figura 3.4: Fluxo da atividade de elaboração do descritivo não funcional.

3.2.2 Produzir um protótipo

Nesta fase é feita a captura das necessidades do usuário relativo às telas do sistema e traduzido no termo de um protótipo. O Protótipo Conceitual é um rascunho de como o “site” irá aparecer, sua navegação, identificação, disposição de quadros e outros detalhes visuais. Ele é desenhado pelo designer e aprovado pelos analistas de requisitos e representantes do usuário, guiará todo o desenho das telas do projeto.

Os recursos necessários à execução desta atividade estão descritos no quadro 3.3.

Quadro 3.3 - Recursos necessários à construção do protótipo.

Papéis	Ferramentas
Analista do Cliente	Entrevistas
conhecedor do Domínio	Descritivo Funcional
Analista	Editor HTML
Designer	Intranet de Documentação

Este atividade inicia com uma reunião marcada pelo analista com o designer e o analista do cliente que tenha o conhecimento do domínio.

Nesta reunião são identificadas as telas a serem construídas e quais são os campos e botões que compõem cada uma delas. Os campos são detalhados em termos de tipo de campo, tipo de informação, se é iniciado com ou sem valor, e os definido valores fictícios para os mesmos. Os botões também são detalhados buscando identificar um nome padrão e para onde o usuário é direcionado quando do seu acionamento. Caso houver necessidade, também deverão ser descritas as regras para validação de campos e navegação de entre telas. Os casos de negócios que fazem com que o sistema seja direcionado ora para um local ora para outro também deverão ser postos às claras, assim como qualquer outro detalhe visual ou negocial.

De posse destas informações, o designer construirá um protótipo, que deverá ser apresentado ao cliente para aprovação, juntamente com o analista. O protótipo deverá ser trabalhado o quanto seja necessário para que não restem dúvidas com relação ao funcionamento do sistema, e que todas as regras de negócio estão sendo contempladas. É necessário que o cliente, através do seu gerente de projeto, dê o aceite no protótipo através de um documento de aceite elaborado pelo analista.

A figura 3.5 apresenta o fluxo desta atividade, onde podemos verificar que o produto desta atividade é o protótipo construído, aceito pelo cliente e disponibilizado na intranet de documentação.

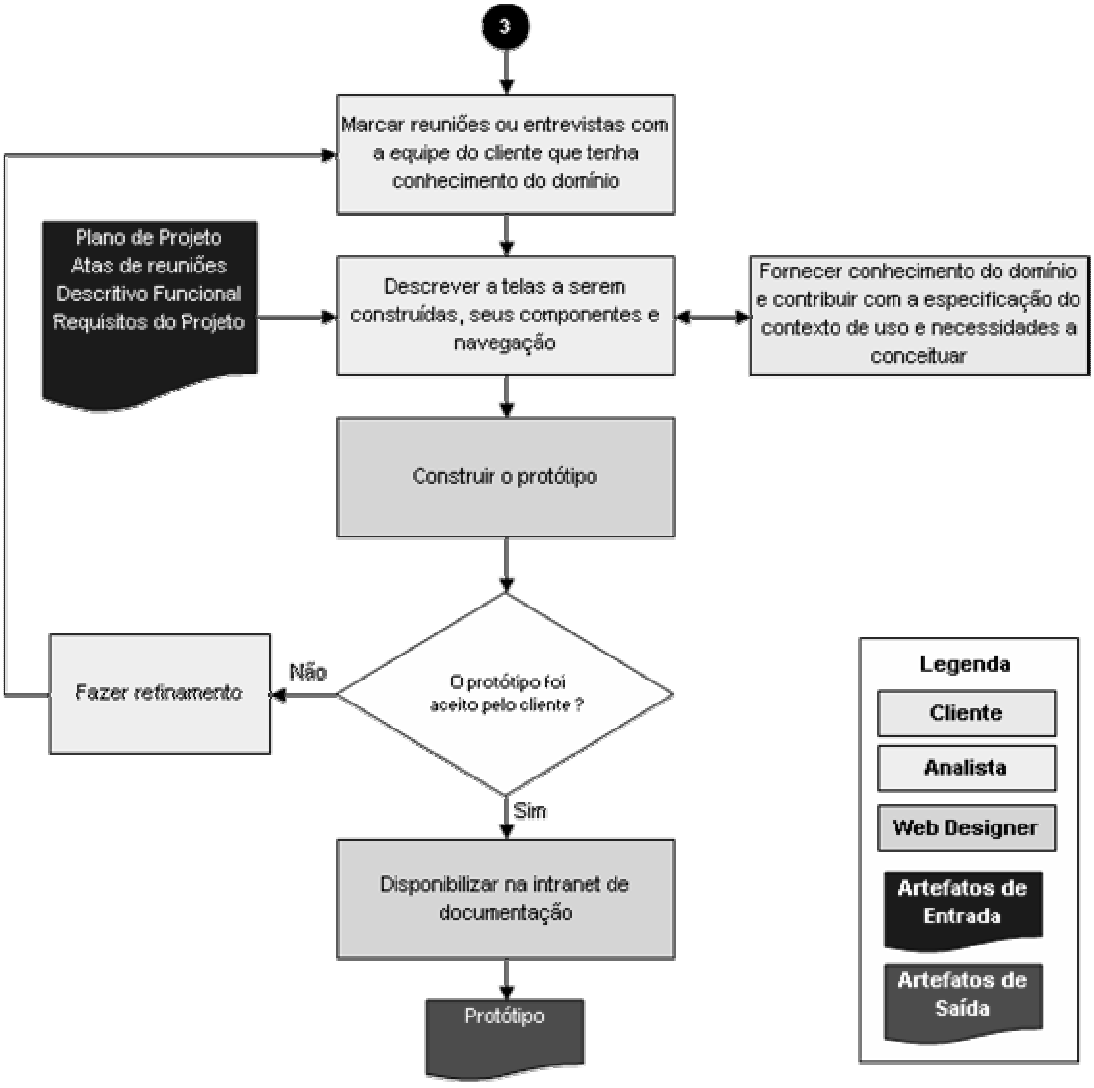

Figura 3.5: Fluxo das atividades de prototipação.

3.2.5 Detalhar o use-case

Após a aprovação do protótipo, é hora de detalhar os use cases. O detalhamento do use-case implica na especificação dos passos definidos no use-case, na definição da interface com atores externos, na construção do diagrama de

seqüência e do diagrama de navegação. É partir destes documentos que o projeto, de fato, será gerado.

Para sua execução basta ao analista seguir o fluxo de eventos e detalhar cada entrada no sistema. Deve ser lembrado que entradas são operações, derivando em transações, e as saídas do sistema resultados, ambos com views equivalente.

Os critérios de entrada são os Descritivo Funcional e Não Funcional concluídos e o Protótipo desenhado. A saída inclui um diagrama de use case com sua especificações, um diagrama de seqüência, um diagrama de navegação e um documento retratando as interfaces utilizadas.

Use Case

Os use cases devem estar contidos no dicionário da ferramenta case e devidamente documentados. Cada use case deve:

- Definir pré e pós condições;
- Descrever o fluxo principal, através uma seqüência básica de transações, na sua forma normal;
- Mostrar os fluxos alternativos definidos através da interação do usuário com o sistema;
- Mostrar os fluxos de exceção tratando situações de erro;

Os recursos necessários à execução do detalhamento do use case estão descritos no quadro 3.4.

Quadro 3.4 - Recursos necessário para o detalhamento das atividades.

Papel	Responsabilidades
Analista do Cliente, conhecedor do Domínio Analista	Intranet de Documentação Storyboard e Protótipo Ferramenta Case

No fluxo dos eventos deve ser incluído como e quando o use case inicia e termina, quando o use case interage com os atores e qual é a troca de informação que há entre ator e use case.

É interessante que seja estabelecido um padrão de identificação que torne o use case único. Um padrão pode ser UCDDDnnn, onde DDD são três letras que indicam o domínio a que pertence o use case, por exemplo, um módulo de vendas pode ser “VND”, e nnn indica o número do use case, – esta identificação deve servir como trilha por todo o processo de desenvolvimento.

Os critérios de entrada deste passo são os Descritivos: Funcional e o Não Funcional, concluídos; e o protótipo.

Assim a seqüência de atividades para criação do use case é a seguinte, o analista inicia criando o módulo de análise na ferramenta case. Em seguida ele define os atores e cria um novo diagrama de use case. A partir do protótipo identificam-se as views, levando em consideração que a cada tela é uma view separada. Devem ser identificadas também as interfaces. Em seguida desenha-se o use case. Na figura 3.6 podemos ver um exemplo de um diagrama de use case.

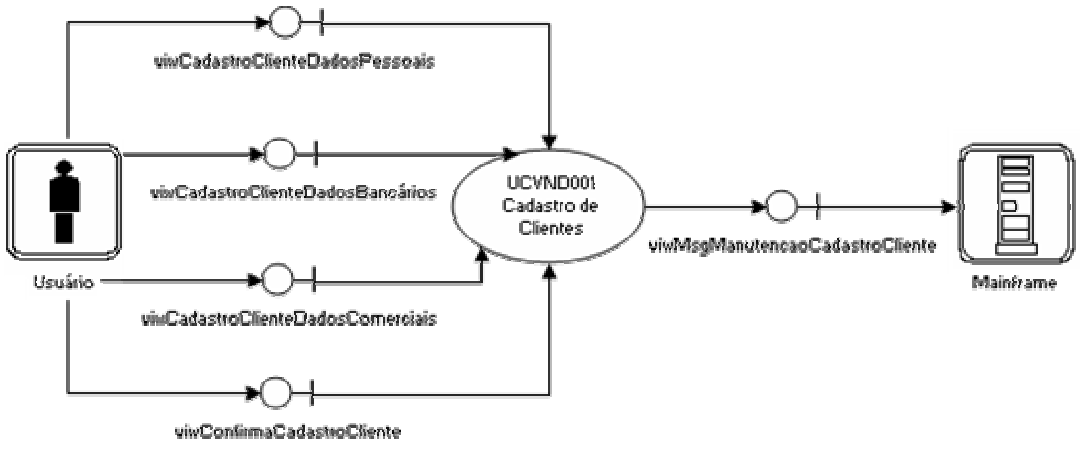

Figura 3.6: Modelo de use case.

Estas views são uma representação da parte visual, e por isso devem ser detalhadas na forma textual. Este detalhamento deverá incluir os seguintes campos:

Nome. Deve ser declarado tal como no protótipo.

Tipo de informação. É o tipo de dado retornado que este campo comportará, por exemplo: Alfanumérico, numérico, data, etc.

Tipo de Campo. Descreve o tipo de campo utilizado, como ComboBox, EditText, Label, Radiobutton, CheckBox, etc.

Tamanho. Define o tamanho que o campo ocupa na tela.

Formato. Define se o campo tem formato pré-definido, como no caso de datas: DD/MM/AAAA, ou moeda: R$ 9.999,99.

Fonte. Define o tipo de fonte a ser utilizada e deve ser preenchido segundo o padrão definido na folha de estilos, como: normal, mas pode também, em casos especiais, definir o nome da fonte, seguido pelo tamanho, cor e se é negrito.

Alinhamento. Define o alinhamento do campo, devendo ser preenchido com: direita, esquerda, centralizado ou justificado.

Ordem de tabulação. Define a ordem de tabulação na qual o campo estará inserido. Deve-se tomar o cuidado de não repetir o número em dois campos.

Entrada/Saída. Define se o campo é de entrada ou saída de dados, devendo ser preenchido com: Entrada ou Saída.

Obrigatório. Deve ser preenchido com Sim ou Não, e define se o campo é obrigatório.

Valor Inicial. Define qual se há e qual é o valor que o campo deverá apresentar quando for iniciado.

Origem. Define a origem dos dados, sendo preenchido com o nome do campo retornado da interface e o nome da própria interface.

Habilitado. Deve ser preenchido com sim ou não e define se o campo deve estar habilitado ou não. Para campos que deverão estar habilitados, mas invisíveis ao usuário, utilizar: Sim, mas não visualmente.

Validação. Descreve as validações que o campo deverá ter, como: <> branco, <> zero, < 100, etc.

Mensagem de erro – define qual mensagem de erro deve ser apresentada de acordo com a sua validação. As mensagens de erro devem estar agrupadas em uma tabela de erros única, e devem ser reutilizadas sempre que possível.

Em seguida, são preenchidos os atributos do Use Case na ferramenta case. Se não houver ferramenta case, os diagramas poderão ser desenhados em uma ferramenta gráfica e um arquivo texto deverá ser criado para cada componente do use case, tomando o cuidado de que o nome do arquivo seja sempre igual ao do componente.

O analista deve iniciar fazendo uma descrição do use case, procurando esclarecer, de forma sucinta, qual funcionalidade este use case implementará. Em seguida ele descreverá as pré-condições para que o use case possa ser iniciado sem problemas, e quais serão os resultados produzidos pelo use case.

O próximo passo é descrever em forma de um fluxo, as operações a serem realizadas pelo usuário e as reações do sistema. A descrição começa a ser feita dentro do atributo Fluxo Principal e conforme as ações tomadas pelo usuário, o fluxo pode ser desviado para os Fluxos Alternativos ou Fluxos de Exceção. O fluxo começa a ser descrito desde a primeira ação do usuário no menu e se estende até a sua conclusão. Cada passo deve ser numerado e cada botão deve gerar um fluxo alternativo.

Boas práticas no detalhamento dos use-cases incluem identificar claramente quais interações pertencem ao sistema e quais ocorrem entre os diversos atores, descrever somente os eventos pertencentes ao use case e omitir terminologia vaga como “por exemplo”, “etc.”, “obter informação”.

Os produtos deste passo são os use-cases construídos na ferramenta case, o protótipo construído e o diagrama de seqüência construído. Todos aprovados pelo usuário.

Interfaces

O analista deve definir junto ao usuário quais e como serão as interfaces com o sistema. Há dois tipos de interface:

- Interfaces com o cliente – mensagens trocadas entre o terminal cliente e o sistema, correspondem as views de entrada e saída com o cliente.

- Interfaces com sistemas legados ou mainframes – layout das mensagens trocadas entre o sistema e sistemas legados ou mainframes.

Os artefatos de entrada deste passo são os Descritivos: Funcional e Não Funcional, concluídos, o Use Cases detalhado e o Protótipo desenhado. Os recursos necessários para execução deste passo estão descritos no quadro 3.5.

Quadro 3.5: Recursos necessários à elaboração do documento de interfaces.

Papel	Responsabilidades
Analista do Cliente, conhecedor do Domínio Analista	Entrevistas Intranet de documentação Editor de Textos

A analista de posse dos use cases, do protótipo, e dos requisitos, marcar entrevistas com o usuário, a fim de identificar os atributos a serem trocados entre os atores em questão. Estes atributos devem estar dispostos em um layout definido pelo tipo de interface: cliente ou mainframe, respeitando os protocolos estabelecidos pela infra-estrutura.

Novamente, são necessárias revisões e o aceite do usuário. As interfaces deverão estar agrupadas, se possível, por módulos, e devem ser reutilizadas em outros módulos sempre que possível. Após o aceito do cliente, o documento deverá ser publicado na intranet de documentação.

Assim, os critérios de saída deste passo é o documento de interfaces construído, revisado, aceito pelo cliente e publicado na intranet de documentação.

Diagramas de Seqüência

Os diagramas de seqüência são ferramentas úteis no esclarecimento do sistema. No caso da análise o diagrama de seqüência descreve as interações entre o sistema e seus atores, isto é, componentes externos ao sistema. Entende-se por componente externo outros sistemas, mainframe e browsers. Isto implica que as views também devem ser consideradas atores.

Tipicamente um diagrama de seqüência tem o formato mostrado na figura 3.7.

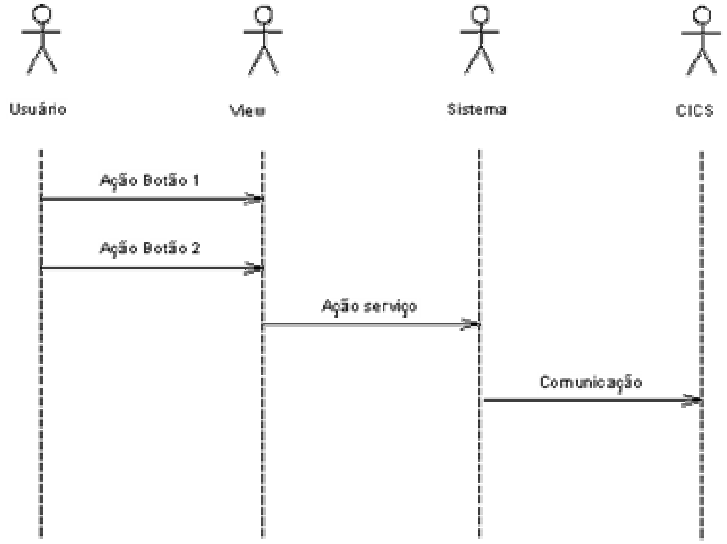

Figura 3.7: Diagrama de Seqüência.

A comunicação entre Usuário e View representa a navegação (interface gráfica), descrito no formato <ação><item>, <ação> pode ser:

- selecionarMenu – seleciona de um Menu um item <item>. Exemplo: selecionarMenuCadastroDeClientes seleciona o item Cadastro do menu citado;
- selecionar<botão> - neste caso qual o botão foi pressionado. Exemplo: selecionarConfirmar indica a pressão no botão Confirmar;
- informar<item> indica quais campos devem ser digitados. Exemplo: informarDadosCadastrais preencher os dados da view dados cadastrais.

Para um bom entendimento do diagrama de seqüência, sua leitura deve ser acompanhada do protótipo.

Diagrama de Navegação

Nesta fase de análise captura-se o comportamento visual, ou seja, captura as necessidades do usuário em termos de navegabilidade do protótipo. A saída inclui um diagrama de navegação. Este diagrama mostra os caminhos possíveis ao usuário quando ele utiliza o sistema. Fortemente baseado no protótipo, deve ter relação com este.

O comportamento visual deve mostrar onde está inserido o módulo – deve haver um diagrama de navegação geral do sistema, uma árvore onde as folhas são os módulos, e uma árvore de acesso: o diagrama de navegação.

O diagrama de navegação corresponde, em UML, ao diagrama de atividades tendo cada atividade como um acesso ao sistema (botão ou seleção de menu) e, suas folhas as páginas ou botões de acesso. Quando não houver direção este acesso será pelo mesmo caminho da árvore.

Os critérios de entrada deste passo são os descritivos: Funcional e Não Funcional, concluído, os use cases definidos e detalhados e o protótipo construído.

Os recursos necessários para a execução deste passo estão descritos no quadro 3.6.

Quadro 3.6 - Recursos necessários à elaboração do diagrama de navegação.

Papel	Responsabilidades
Analista	Diagrama de Use Case e especificações Storyboard e Protótipo Ferramenta Case

Os passos recomendados para a construção de um *diagrama de navegação* são os seguintes: identificar onde se encaixa o módulo dentro do sistema, identificar os caminhos principais e os alternativos do protótipo (incluindo botões e retornos), e por fim desenvolver o diagrama de navegação.

O diagrama de navegação deve conter todas as telas do módulo e as ações que podem ser executadas. “Ações” podem ser a escolha de um item de menu, o pressionamento de um botão de ação, o pressionamento de um botão seletivo

(rádio, seleção), a escolha de uma lista de escolha ou o pressionamento de um enter implícito.

O diagrama deverá representar ainda, condições lógicas, como por exemplo: botão confirma pressionado e pessoa jurídica escolhida, ou botão confirma pressionado e pessoa física escolhida.

Para representar este diagrama na UML, deve ser utilizado o diagrama de navegação, conforme a descrito na figura 3.8.

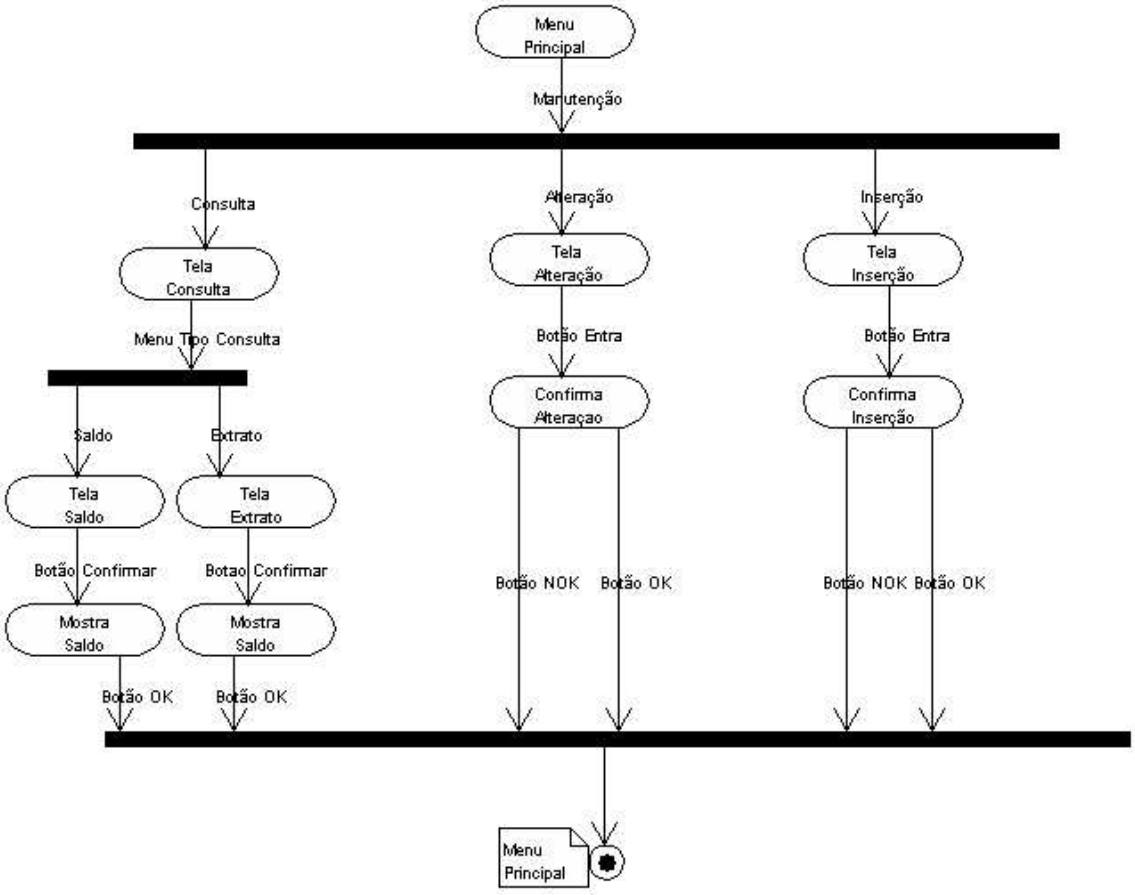

Figura 3.8: Exemplo de diagrama de navegação.

Durante a construção do diagrama algumas regras devem ser seguidas, como:

1) Todo menu deve começar com a palavra menu e ser representado por uma barra de junção de atividades.

2) Todo item de menu deve estar no formato <nome do item> em minúscula com a letra de aceso rápido em maiúsculas.

3) Todo botão deve ser representado por uma seta de fluxo com apenas seu nome.

4) Toda Seleção deve ser representada por uma seta de fluxo com <nome da seleção>.item selecionado.

5) Toda tela deve ser representada por uma atividade e ter seu nome como o da atividade.

6) Telas modais devem ser hachurradas.

7) O retorno padrão (de volta às telas originais) não necessita ser especificado.

Situações de decisão devem ser representadas pela decisão no UML.

Os critérios de saídas deste passo são o Diagrama de navegação construído.

Todos os artefatos descritos acima deverão estar aprovados e aceitos pelo usuário, sendo em seguida publicados na intranet de documentação.

3.2.6 Produzir o Modelo Objeto

Para descrição de um objeto, e para que um sistema possa ser visto como um objeto é necessário uma visão, dita, estática deste. Pode-se considerá-la uma "fotografia" ou como já se usou dizer um "instantâneo". Esta estrutura é descrita pelo Modelo de Objeto, conforme definido no processo de desenvolvimento FUSION.

Embora a descrição do processo mostre a descoberta da estrutura e, após, do comportamento, a estrutura real não prescinde deste último, aliás, engloba-o. A relação entre os dois processos (e documentos) não é linear, mas, uma rede em que há influências mútuas, não podendo dar-se como encerrada a estrutura sem

conhecer o comportamento e vice-versa. Isto posto, a confecção de ambos depende de cada um e deve ser altamente incremental: a partir do comportamento (use case), constrói-se o Modelo de Objetos; a partir do MO, revisa-se as seqüências e volta-se ao MO e assim por diante.

Para aumentar esta rede de influências o projeto visual também efetua alterações em ambos e recebe alterações. O analista, se não tiver consciência disto, corre o risco de deixar incompleto ou incongruente sua descrição.

Os critérios de entrada deste passo são os descritivos: Funcional e não Funcional, concluídos, os Use Cases definidos e o Protótipo desenhado.

Os recursos necessários para o cumprimento deste passo estão descritos no quadro 3.7.

Quadro 3.7 - Recursos necessários para a elaboração do modelo de objeto.

Papel	Ferramentas
Analista	Protótipo Diagrama de use case Intranet de documentação

Para executar a construção do modelo de objetos são necessários os seguintes passos: Obtêm-se os Use Cases, Protótipo, e Descritivos. Identificam-se as classes candidatas ao modelo de objetos, classificando-as. Identificam-se os relacionamentos entre as classes. Delimita-se a fronteira externa no sistema, definindo quais classes candidatas pertencem ao Modelo do Sistema Objeto (classes externas a ele são atores nos UC e relacionamentos que cruzam esta fronteira são operações do UC). Para estas classes seus atributos correspondem a views de entrada ou saída. Para as classes internas é necessário definir os atributos. Enfim desenha-se um novo diagrama de objetos na ferramenta case, criando as classes, relacionamentos e atributos. A nomenclatura de classes e atributos deve seguir os padrões definidos pelo cliente.

Deve-se então revisar o modelo com o usuário e refiná-lo tendo em comparação as especificações e os Use Case, devendo o cliente, dar o aceite deste documento quando da sua aprovação.

Os critério de saída deste passo é o Modelo Objeto construído, aceito e publicado na intranet de documentação.

3.2.7 Fazer o plano de testes

O processo de testes proposto englobará os seguintes tipos de testes:

- Teste Unitário;
- Teste Integrado;
- Teste de Sistema;
- Teste de Aceitação.

Teste Unitário - São testes básicos realizados em uma unidade do software, testando uma parte específica do processo de negócio, da aplicação e/ou da configuração. Os testes unitários asseguram que cada caminho de um processo do negócio execute exatamente as especificações documentadas, e contenha entradas claramente definidas e resultados previstos.

Teste Integrado - O Teste Integrado é projetado para testar os componentes do software de forma integrada para verificar se eles realmente executam como um programa. O Teste de Integração demonstra que todos os componentes já testados unitariamente, quando integrados, funcionam corretamente e consistentemente. O objetivo principal é testar a combinação dos componentes, expondo-a aos fatores críticos dessa integração. Deve ser desenvolvido pelo projetista.

Teste de Sistema - O Teste de Sistema assegura que o sistema de software integrado satisfaça os requisitos da especificação por completo. Testa a configuração para garantir resultados conhecidos e previstos. Deve ser desenvolvido pelo analista.

Teste de Aceitação - O Teste de Aceitação é realizado em ambiente idêntico ao de produção. Este teste é desenvolvido e realizado pelo cliente para

determinar quando o sistema satisfaz seus critérios de aceitação. Baseado nos resultados do teste de aceitação, o cliente determina quando aceitar ou rejeitar o sistema. Deve ser desenvolvido pelo analista e aprovado pelo cliente.

A execução do teste deverá terminar em uma das duas condições:

- **Normal**: todos os procedimentos do teste (ou script) são executados como previsto e de forma completa.
- **Anormal ou prematuro**: os procedimentos do teste (ou script) não são executados completamente ou como previsto. A causa do término anormal/prematuro deve ser identificada, corrigida, e o teste deve ser executado novamente, antes de qualquer atividade de teste adicional ser realizada.

3.3 Intranet de documentação

A análise é uma atividade, sobretudo, de comunicação. Isto posto, é muito importante que os receptores de sua mensagem possam apreender seu significado e, para tanto, é essencial que este aprendizado seja facilitado através desta atividade.

Nem sempre os projetistas e integradores (aqueles que recebem informação da análise) possuem acesso à ferramenta case. Muitas vezes, também, a informação está segmentada em diversos locais, de modo a facilitar o aparecimento de erros durante a fase de projeto.

Considerando isso, é necessário que se crie uma intranet de documentação, que nada mais é que uma junção de diversos artefatos em um único local. Os artefatos que compõem a intranet de documentação são:

- Descritivo Funcional;
- Descritivo Não Funcional (FLURPS);
- Análise Estrutural: Modelo Objeto;

- Análise Comportamental: Diagramas de Use Case, Diagramas de Seqüência e Diagramas de Navegação;
- Interfaces;
- Protótipo.

Os recursos necessários para a execução deste passo estão descritos no quadro 3.8.

Quadro 3.8: Recursos necessários à publicação na intranet de documentação.

Papéis	Ferramentas
Analista Designer	Editor de Textos com suporte a html ou Editor Html Editor Gráfico

O trabalho de publicação pode ser executado pelo analista ou pelo designer. O sistema deve ter uma página principal com os links para todos os módulos. Cada módulo deverá conter uma página inicial com os links para todos os documentos produzidos na análise. Estes documentos deverão estar no formato html e sempre que fizerem referência a outro documento deve-se incluir aí, um link para o mesmo. Todos os documentos devem ter um link para o menu principal do sistema e do módulo a que pertence.

Os diagramas devem ser transformados em imagens, no formato “JPG” por exemplo, e inserido no contextos dos seus documentos html.

Os critérios de saída deste passo são o módulo publicado e acessível na intranet, contendo os seguintes artefatos representados: Descritivo Funcional e Não Funcional (FLURPS), Modelo Objeto, Use Cases, Diagramas de Use Case, Protótipo e Documentos de Interfaces.

3.4 Documentos de Inspeção, Controle e Aceite

É necessário que existam documentos que permitam controlar o processo. Nesta proposta os documentos são classificados nas seguintes categorias:

1. Inspeção e Auditoria – os Review Docs permitem que haja uma revisão do documento e um aceite formal.

2. Aceite – aceitação de fases, ou outros eventos.

3. Controle – finalização ou inicia passagem de atividades.

Veja a seguir o quadro 3.9 com estes documentos.

Quadro 3.9 – Documentos que permitem controlar o processo.

Documento	Tipo	Objetivo	Origem/Aprovação	Destino
Iniciação de Módulo	3	Dar inicio a uma atividade de desenvolvimento do módulo, cuja obtenção dos requisitos é a inicial (TDR).	Gerente do projeto	Analista Responsável
Convocação de equipe	3	Convida os envolvidos a fazerem parte do projeto	Analista, gerente	Usuário Analista de domínio Prototipador Facilitador
Descritivo Funcional	1,2			
Requisitos	1,2			
Protótipo	1,2	Aprova o Protótipo Conceitual	Análise, Design Auditor	Projeto
Use Cases e Diagramas de Seqüência	1,2			
Modelo de Objetos	1			
DUA	1			
Interfaces	1,2			

3.5 Demais fases

As fases de projeto, implementação e testes, não fazem parte do foco deste trabalho. Sabe-se que o projeto e a implementação dependem muito da estrutura definida como: linguagem de programação, comunicação entre as entidades, infra-

estrutura de sistema e outras inúmeras variáveis, e não caberia aqui, um detalhamento que atendesse apenas uma parcela dos leitores.

3.6 Fábrica de software

Através de uma Fábrica de Software, a WOPM Informática pouparia o cliente de todo um ambiente de produção, envolvendo recursos técnicos e humanos. O uso de uma linha de montagem de software implica numa elaboração mais rápida e disciplinada que, além de promover a redução de custos, aumenta a produtividade, contribuindo para o cumprimento de prazos. A qualidade é um outro benefício inerente a este processo, uma vez que a Fábrica possui todo um controle sobre as atividades e resultados a serem atingidos.

O conceito de Fábrica de Software já tem uma longa história. Desde o início de 1990 vários projetos foram implementados utilizando esta estrutura.

A maturidade e otimização dos processos, aliada aos softwares de acompanhamento e gerenciamento, a preocupação constante em aprimorar os conhecimentos da equipe, proporcionando ao time treinamentos na sua área de atuação, e a competência das pessoas que nela trabalham são os diferenciais para que uma empresa de desenvolvimento de software seja respeitada e reconhecida.

3.7 Considerações finais

Para que o processo de análise mereça o devido crédito, seus documentos devem refletir a realidade do sistema, para que possam servir de base para um bom projeto. O projetista ao ler os documentos de análise, não deve ter dúvidas quanto ao funcionamento do sistema e seus requisitos.

A proposta descrita neste capítulo tem como objetivo estabelecer um processo que permita a geração de documentos sólidos e confiáveis.

Outra decisão seria propor uma melhoria continua dos processos, propondo-se medidas corretivas que melhorem o desempenho desses processos, tornando a análise ainda melhor e esclarecedora.

O próximo capítulo descreve um estudo de caso aplicado na WOPM Informática, tendo como base a implantação da proposta de processo de análise, e buscando medir e avaliar o desempenho a partir de sua conclusão.

4 APLICAÇÃO PRÁTICA DA METODOLOGIA

4.1 A Empresa WOPM

A WOPM Informática é uma integradora de negócios voltados para a Internet, buscando sempre a solução que agregue mais valor ao negócio do cliente. Ela executa projetos de desenvolvimento de sites e softwares baseado na Internet, primando pela qualidade e satisfação do cliente.

Missão da empresa - Prestar serviços que permitam ao cliente fazer uma análise das vantagens das novas tecnologias Internet, e que transformem seu negócio, fazendo com que o cliente prospere na nova economia digital.

4.2 Histórico

Fundada em 1988, sua sede corporativa está sediada na cidade de Cascavel, estado do Paraná, no Brasil. Tendo como diretor o Sr. Olavo José Luiz Jr.

Hoje, a principal estratégia da WOPM para a Internet é a produção de sistemas voltados para a Internet. Esta estratégia é decorrente da convicção de que a Internet evoluirá de uma coleção de sites web, para um mercado virtual de serviços baseados na Internet que poderão ser invocados a partir de qualquer dispositivo. A WOPM pretende acompanhar essa evolução da Internet, impulsionando a criação dessa nova geração de serviços, necessários para suportar, sem falhas, um mundo no qual bilhões de dispositivos estarão gerando trilhões de transações.

4.3 Organização

Hoje a empresa se divide em duas áreas: serviços de conectividade e desenvolvimento, contando com 15 funcionários distribuídos em 3 escritórios.

Os serviços de conectividade englobam: o provedor de Internet, com a venda de acesso discado e privado, a montagem de redes corporativas remotas utilizando as tecnologias de ponta disponíveis no mercado, e a venda de equipamentos que advenha destes serviços. Esta área conta com 9 pessoas, sendo 2 instaladores de redes, 1 técnico em comunicações, 1 administrador do provedor, 2 vendedores.

A área de desenvolvimento está preocupada com a produção de sites e sistemas voltados para a web. Para isso, a área conta com uma equipe composta de: 2 designers, 1 analista, 2 programadores, 1 analista/administrador de banco de dados.

A empresa, como toda empresa, possui uma área administrativa, que conta hoje com 3 pessoas, incluindo o diretor.

4.4 O desenvolvimento de software na empresa

A área de desenvolvimento de software da empresa iniciou suas atividades no final da década de 80. Os softwares eram feitos de maneira artesanal, contando apenas com 2 profissionais, que se alternavam nas atividades de análise e programação. A linguagem de programação utilizada na época era o Clipper.

Na época não eram utilizadas técnicas ou ferramentas de análise, embora seguisse um processo: entrevista com o cliente, reunião da equipe para discussão da solução, programação da solução, apresentação para o cliente, correção de erros, aplicação de melhorias, implantação e manutenção.

Esta situação durou cerca de 4 anos, quando em 1993, por mudança da estratégia da empresa, a área de desenvolvimento foi desativada.

Após 5 anos, com o advento da Internet, a empresa reiniciou novamente suas atividades de desenvolvimento. Desta vez, o foco era a produção de sites. O setor foi organizado pelo diretor da empresa, que instituiu um processo de desenvolvimento bem parecido com que ele trabalhava no início da empresa: entrevista com o cliente, apresentação de um protótipo, construção do site sob o

esqueleto do protótipo, apresentação para o cliente, correção de erros, implementação de melhorias, publicação do site e manutenção.

Este processo se mostrou eficiente tanto na época da programação em Clipper, quanto agora com a produção de sites empresarias e pessoais, satisfazendo os clientes tanto em prazo quanto em qualidade.

A alta competitividade fez surgir nas empresas um grande necessidade de geração rápida de informações que pudessem ajudá-las a tomar decisões importantes em seus negócios. Esta necessidade fez com que a Internet evoluísse de uma coleção de sites para uma ferramenta estratégica de comunicação entre clientes e fornecedores.

Com esta evolução e a melhoria das comunicações, tanto na qualidade dos serviços, quanto nos preços, fez surgir um novo horizonte para as empresas, onde passou a ser permitido a construção integral de sistemas baseados na Internet.

A WOPM Informática procurando atender aos anseios de seus clientes, iniciou as atividades de desenvolvimento de sistemas baseados na web. Inicialmente, apenas com a construção de sistemas que extraíam informações dos bancos de dados e os disponibilizavam para clientes, fornecedores e tomadores de decisões, ou seja, a parte gerencial do sistema. A equipe de desenvolvimento foi composta dos designer já existentes, mais 1 analista e 1 programador.

Num segundo momento, passou a desenvolver também a parte operacional do sistema. Para que isto fosse possível houve a necessidade da ampliação do quadro de profissionais, embora esta ampliação tenha sido feita de forma desordenada, conforme a necessidade. Hoje a equipe, como já foi dito anteriormente, é composta de 2 designers, 1 analista, 2 programadores, 1 analista/administrador de banco de dados.

Também é válido salientar que a empresa quando iniciou suas atividades, as metodologias de desenvolvimentos existentes eram estruturadas. Durante o período em que a área de desenvolvimento de software da empresa esteve desativada, surgiu um novo paradigma, a orientação a objetos. Este novo paradigma foi estabelecido como padrão de mercado através de novas linguagens de

programação, metodologias e ferramentas de desenvolvimento. Neste sentido, a empresa perdeu valiosos anos de experimentação, sendo que o envolvimento de profissionais que transcenderam a barreira entre os dois paradigmas, prejudicaram a empresa no estabelecimento de um processo de produção de software.

4.5 Metodologia utilizada

Com o objetivo de traçar um parâmetro entre o processo de desenvolvimento antes e depois, durante o tempo de produção de um módulo de sistema, que foi de três meses, foram acompanhadas as atividades de produção de software da empresa, buscando fazer um mapeamento das atividades desenvolvidas durante o processo.

Estas atividades foram registradas pelos próprios profissionais através de um formulário intitulado “Formulário de acompanhamento de atividades”, conforme o Anexo I, distribuído aos mesmos através de uma planilha compartilhada, onde todos tinham acesso.

O formulário buscava, além de registrar as atividades, quantificar o tempo gasto com cada uma, documentar as ferramentas utilizadas, artefatos de entrada e os artefatos de saída.

A equipe foi reunida e uma explanação foi feita acerca do formulário, seu objetivo e forma de preenchimento.

Com o andamento das atividades, os formulários foram analisados semanalmente buscando classificar as atividades e estabelecer uma ordem para as mesmas, com o intuito de enquadrá-las em um processo.

Reuniões mensais foram realizadas a fim de informar a equipe sobre a evolução da pesquisa e esclarecer dúvidas surgidas na análise dos formulários.

Ao final, foi feita uma análise dos dados recolhidos, buscando inicialmente definir o modelo de processo utilizado pela empresa. Logo em seguida buscou-se,

através de uma análise de pareto, descobrir quais eram as atividades que mais consumiram tempo durante a fase de análise e sua principal causa.

O resultado deste trabalho está descrito a seguir.

4.6 Descrição de um processo de construção

A empresa foi contratada para criar uma interface web para um sistema CICS. De forma sucinta o processo de desenvolvimento é descrito nos próximos parágrafos

O desenvolvimento iniciou-se com a assinatura do contrato. Contrato este que dá uma breve descrição do sistema, os prazos para entrega e os valores. Ao todo, seriam construídos 14 módulos, cada um com sua particularidade e grau de dificuldade. O contrato também previa os requisitos gerais do sistema como desempenho, tecnologia, segurança além de requisitos operacionais, como a necessidade de geração de logs de auditoria para todas as transações de negócio.

A equipe constituída para este projeto era composta pelos seguintes integrantes: 2 analistas de sistemas, sendo que 1 desempenhava o papel de coordenador, 1 programador HTML/JavaScript, 1 programador Java/C, 2 programadores JSP.

A partir daí marcou-se uma reunião com o usuário para coletar os dados necessários para a construção das telas. A reunião foi conduzida informalmente e os produtos foram: um desenho manual das telas, uma descrição textual das funcionalidades de cada tela, uma lista dos serviços CICS com os campos da tabelas a serem consultadas ou alteradas. Com base nos rascunhos, preparou-se uma ata da reunião, onde, posteriormente, foram colhidas as assinaturas do cliente e do analista.

Paralelamente a esta atividade o programador C produzia os serviços para a que comunicação de dados entre os servidores web e o mainframe fosse possível. Foi utilizado o protocolo APPC (Advance Peer to Peer Comunications) - desenvolvido pela IBM e disponibilizado pelo cliente.

Com base na ata, os programadores procuravam desenhar as classes Java. Por vezes, buscavam a ajuda do analista para melhor compreensão da ata, e o analista por sua vez buscava ajuda junto ao cliente.

Com o total entendimento da ata, o programador Java produzia as classes Java, em alguns casos, solicitando novos serviços ao programador C, e os programadores JSP e Html/JavasScript produziam as páginas HTML e JSP.

Os programadores faziam os testes unitários. Baseando-se sempre na ata, o programador e o analista testavam a aplicação produzida. Os erros e melhorias encontrados eram implementados imediatamente, e posteriormente eram feitos novos teste para verificar a implementação da solução. Freqüentemente estes testes eram feitos repetidas vezes.

O módulo pronto era entregue ao cliente que instalava o módulo em um ambiente de testes, onde o ambiente de produção era reproduzido fielmente. Os testes eram feitos com usuários tirados da produção, que pelo histórico das atividades de testes, não conseguiam testar 100% da aplicação, em função de erros impeditivos para a continuação da aplicação.

Os erros encontrados eram enviados ao analista que fazia uma análise, classificando-os em erros e melhorias, segundo ata assinada pelo cliente. Os erros eram corrigidos de forma imediata pelo programador, mas muitas vezes, sem a atualização da documentação de análise ou dos projetos.

Vale salientar que os erros encontrados pelos usuários/testadores eram enviados por e-mail para os programadores que efetuavam os testes. As discussões em função de e-mails perdidos ou não enviados eram freqüentes.

Neste ponto, foram encontrados registros de reuniões para discussão da implementação das melhorias que o cliente julgava imprescindível ao sistema, mas que não foram implementadas devido à falha na análise, que passaram desapercebidas tanto pelo cliente quanto pelo analista.

O resultado destas reuniões eram acordos firmados entre as partes para que a solução fosse implementada. Na pesquisa de satisfação do cliente quanto ao

módulo produzido, um dos pontos levantados foi o valor final pago pelo software, que foi 2 vezes o que foi calculado inicialmente.

Ainda segundo a pesquisa com o cliente, os principais motivos para que o resultado final não fosse satisfatório foram: muitos erros, necessidade de novas funcionalidades não implementadas ou até mesmo não descritas pelo usuário.

Por fim, feita uma análise financeira do software produzido, descobriu-se que ambas as partes não ficaram satisfeitas. Para a empresa o software não trouxe os lucros calculados inicialmente, e para o cliente as renegociações durante o projetos acabaram por prejudicar seu orçamento anual.

Os números finais da produção do software foram:

- Foram produzidas 33 telas;
- Custo total de produção: R$115.000;
- Atraso de 35 dias pelo prazo inicial;
- Encontrados 495 erros;
- Implementadas 35 novas melhorias;

4.7 Análise dos dados

Inicialmente, as atividades da planilha formulário foram classificadas em: análise, projeto, construção e testes. Esta classificação foi baseada nos seguintes conceitos:

- Análise – toda atividade envolvida na elucidação do comportamento do sistema e que resultaram em descritivos, modelos ou diagramas.
- Projeto – toda atividade envolvida na representação das operações do sistema através de modelos ou diagramas.

- Construção – toda atividade envolvida com a codificação do projeto em uma linguagem de programação.

- Testes – toda a atividade envolvida na validação do resultado da construção pelos artefatos da análise.

Em paralelo com a classificação de atividades, procurou-se estabelecer qual era o processo seguido durante a construção de cada módulo. O resultado deste trabalho pode ser visto na figura 4.1

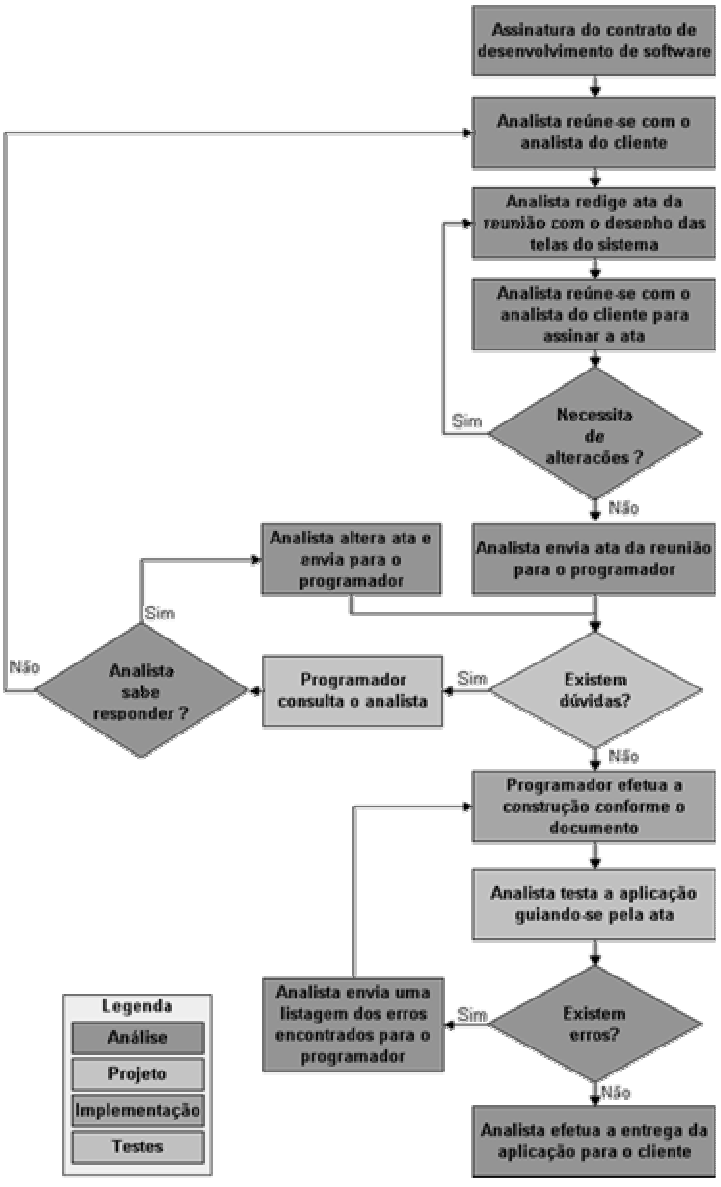

Figura 4.1: O processo de desenvolvimento da WOPM Informática.

Fora alguns casos específicos, como atividades envolvendo a resolução de problemas de ambiente de desenvolvimento, testes e outras, que se caracterizam por atividades de gerência de escritório, o processo descrito na figura 2 não pode ser considerado como um espelho do processo da empresa.

De acordo com a classificação feita, fez-se um somatório dos tempos gastos com as atividades de cada fase, obteve-se resultados interessantes. Como se pode ver na figura 4.2, as atividades de implementação e testes foram as que mais gastaram tempo na implementação.

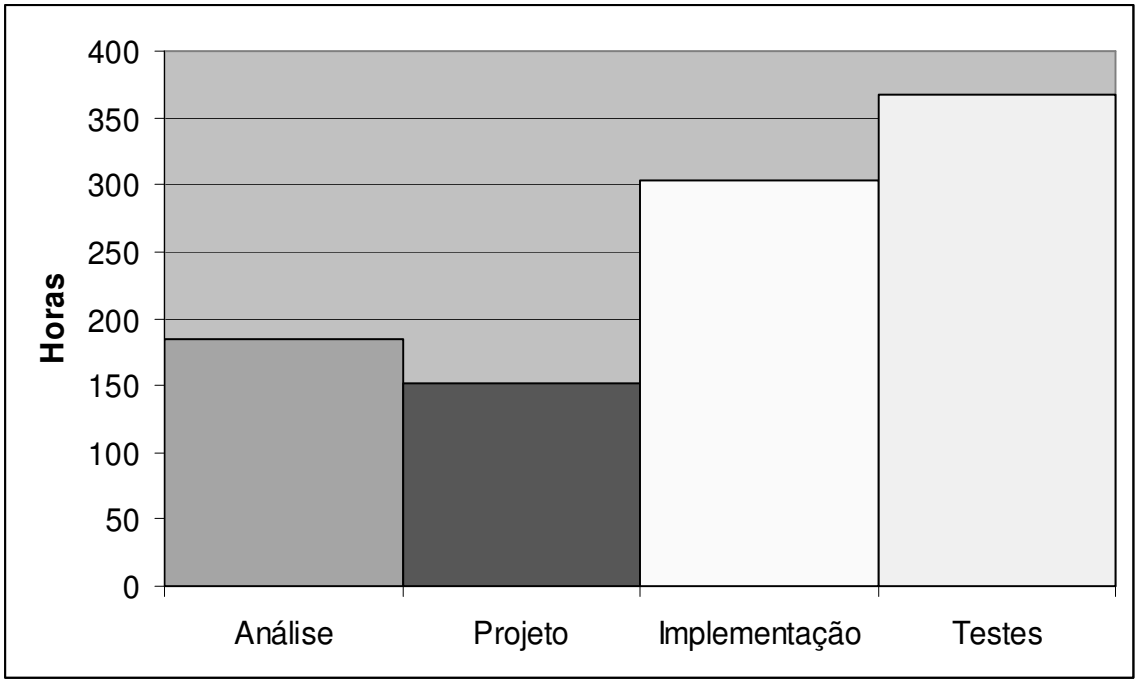

Figura 4.2: Gráfico dos tempos gastos em cada fase do processo.

Pela análise simplória deste gráfico podemos concluir que as atividades de implementação e testes foram as responsáveis pelo atraso no prazo de entrega da aplicação, já que juntas elas consomem 67% do tempo gasto na construção. Mas levando em consideração que, normalmente, a fase de análise é a que gasta mais tempo durante o desenvolvimento, surge a pergunta: que atividades foram realizadas no processo de implementação e testes ?

Para responder esta pergunta, foram analisadas as atividades relacionadas à implementação e testes que estavam descritas na planilha formulário e classificadas em:

- Construção de um novo módulo – construção com base na ata da reunião realizada entre o analista e o cliente.

- Correção de erros – correção de erros com base na lista de erros encontrados pelo analista nos testes.

- Implementação de melhorias – implementação de melhorias com base na ata da reunião incrementada pelo analista após os testes.

O resultado desta classificação está descrito na figura 4.3, onde podemos ver um gráfico que representa os tempos gastos com cada atividade, trazendo a informação de que as implementações relacionadas à correção de erros são as que efetivamente gastam mais tempo.

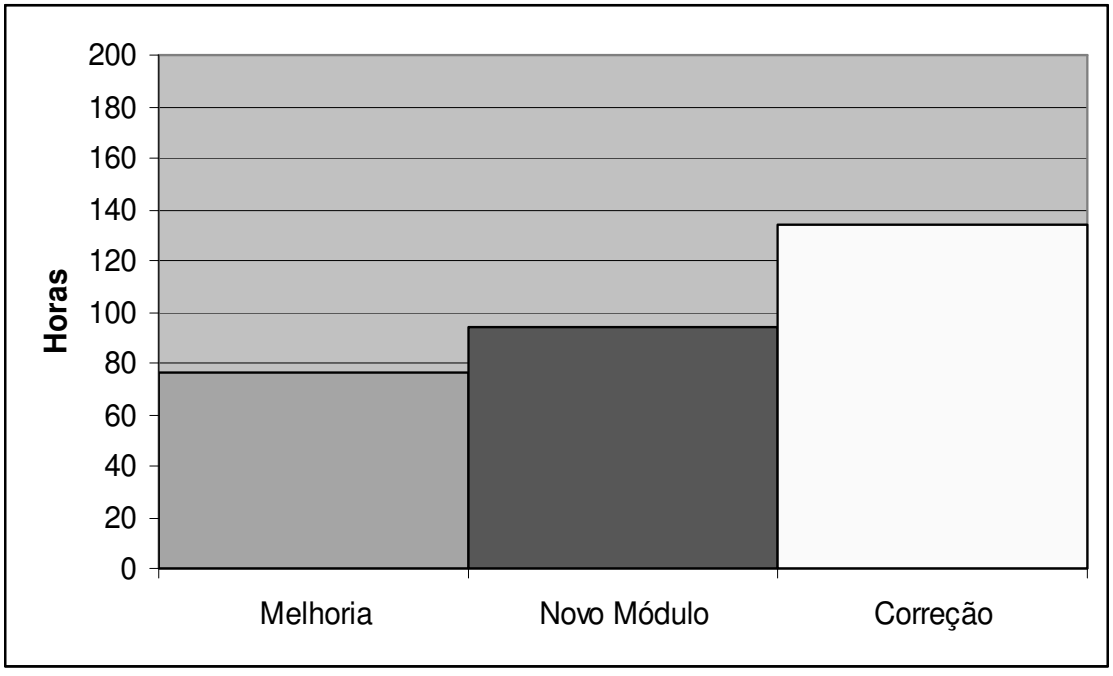

Figura 4.3: Gráfico dos tempos gastos com atividades de implementação.

Em seguida temos a classificação dos testes, que seguiu os seguintes critérios:

- Novos módulos – que englobam todas as atividades relacionadas a testes feitos módulos novos.

- Erros – que envolvem as atividades de testes de correções efetuadas no sistema.

- Melhorias – que envolvem as atividades referente a testes de implementações de melhorias.

O resultado pode ser visto na figura 4.4, onde podemos observar que os erros são os responsáveis por 44% do tempo com testes.

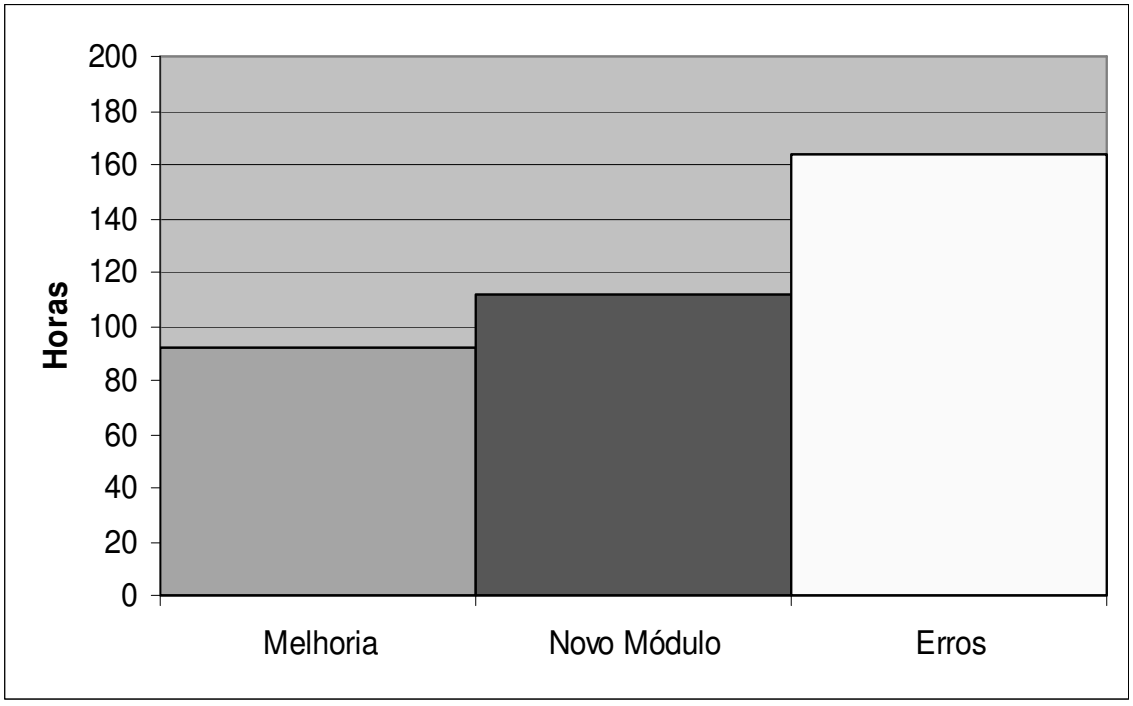

Figura 4.4: Gráfico dos tempos gastos com atividades de testes.

A fim de aprofundarmos a questão dos erros, procuramos saber tanto na implementação, quanto nos testes, se os erros eram provenientes de melhorias ou não. Podemos observar nas figuras 4.5 e 4.6, que não há diferença significativa no gasto de tempo entre atividades de implementação e testes relacionadas com correções de erros e implementação de melhorias.

Assim conclui-se que a grande responsável pelo atraso foi a implementação de melhorias que buscavam “completar” o software para que o cliente desse o aceite para cada módulo. Esta conclusão denota uma falha grande na parte de análise do processo. Os artefatos desenvolvidos hoje pelo analista não estão suficientemente detalhados e não contêm uma visão completa do sistema, prejudicando o trabalho

de implementação, além de causar a geração de melhorias que consomem muito tempo para ser implementadas.

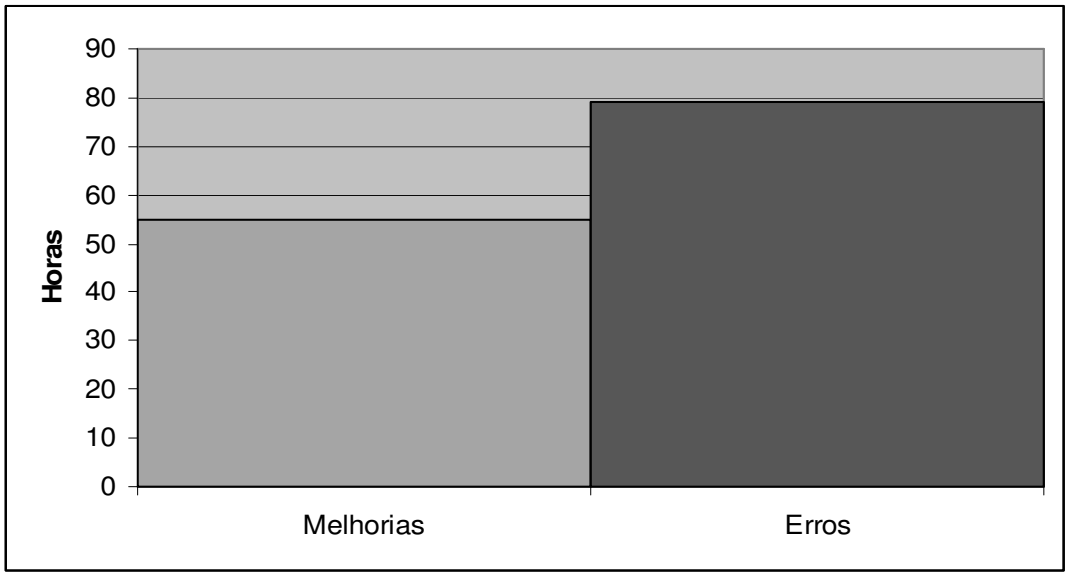

Figura 4.5: Gráfico dos tempos gastos com implementação de melhorias e correção de erros.

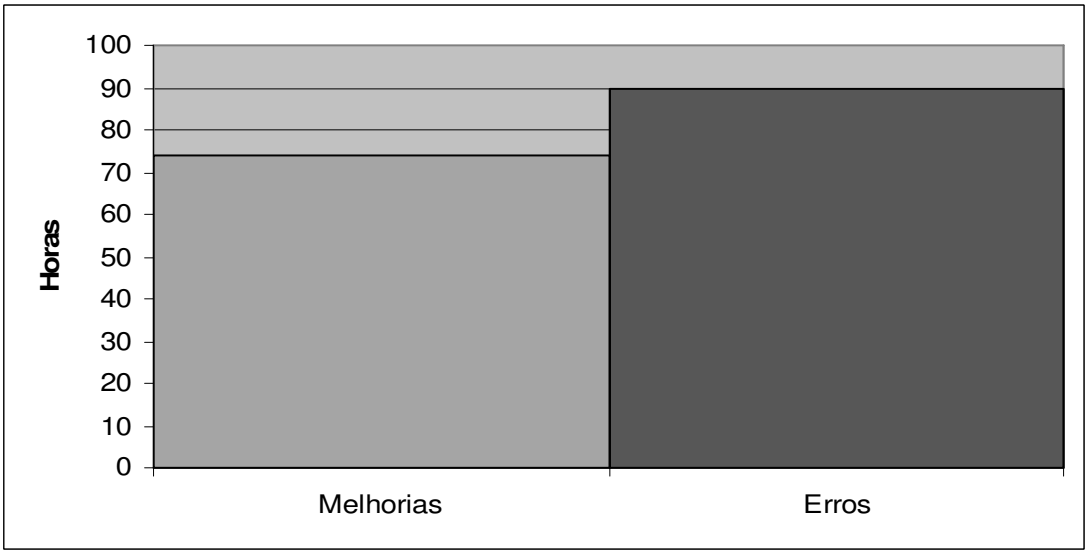

Figura 4.6: Gráfico dos tempos gastos com testes de melhorias implementadas e de erros corrigidos.

Através desta conclusão foi possível elaborar as seguintes perguntas: “Quais são os pontos falhos da análise que estão provocando um número tão elevando de melhorias ?”

Através de uma entrevista com cada integrante da equipe, onde foram apresentados os resultados da análise do formulário, foi possível levantar uma série de fatores relacionados à pergunta formulada no parágrafo anterior. Podemos destacar as seguintes:

- A ata produzida pelo analista não contém muitos detalhes;
- Não é feito um protótipo;
- Faltam artefatos que traduzam de forma clara e precisa o que e quando o sistema deve executar as operações;
- Falta uma definição de como e através de qual documento os testes devem ser executados;
- Várias versões da ata circulam no projeto;
- Não existem ferramentas case.

4.8 Aplicação da Metodologia

Daqui em diante apresenta-se uma aplicação prática do processo de análise, descrito no capítulo 3, na construção do módulo de sistema, pela empresa WOPM Informática Ltda.

Com base na construção de um novo módulo de sistema, para um cliente, procurou-se verificar os efeitos da utilização do método de análise proposto.

4.8.1 Preparação dos envolvidos

A WOPM Informática possui vários projetos em andamento. A aplicação do modelo proposto nestes projetos seria muito custosa e implicaria no aumento dos prazos, devido à necessidade de treinar os profissionais envolvidos.

Desta forma, nesta etapa inicial, realizou-se uma reunião com a direção da empresa e com os profissionais envolvidos com a construção do novo módulo, para explicar-lhes a metodologia e para que entendessem o escopo do trabalho, colaborando assim nos momentos de coleta de dados e informações complementares que pudessem ser úteis para uma melhor análise dos processos.

4.8.2 Metodologia utilizada

A metodologia será praticamente a mesma descrita no capítulo 2, ou seja, será feito um acompanhamento das atividades realizadas durante todo o processo de desenvolvimento do software.

Este acompanhamento, também será através de uma planilha eletrônica compartilhada, intitulada “Formulário de Acompanhamento de Atividades” e o seu preenchimento ficará a cargo dos próprios profissionais envolvidos no processo. O formulário é o mesmo da primeira fase, podendo ser visto no Anexo I.

Da mesma forma que antes, o formulário serviu para registrar as atividades, quantificar o tempo gasto com cada uma, documentar as ferramentas utilizadas, artefatos de entrada e os artefatos de saída.

A análise dos formulários foi feita semanalmente e reuniões mensais foram realizadas a fim de informar a equipe sobre a evolução da pesquisa e esclarecer dúvidas surgidas na análise dos formulários.

Ao final, buscou-se fazer uma análise comparativa entre o antes e o depois, visando descobrir se a metodologia obteve o resultado esperado.

O resultado deste trabalho está descrito a seguir.

4.8.3 Caso Estudado

A empresa foi contratada para a construção de mais um módulo para o sistema web desenvolvido na primeira fase.

O módulo seria constituído de 11 itens de menu, onde o número de casos de uso foi estimado em 22. O contrato assinado previa os requisitos gerais do sistema como desempenho, tecnologia, segurança, além de requisitos operacionais, como a necessidade de geração de logs de auditoria para todas as transações de negócio. O desenvolvimento foi orçado em US$ 13.800 e o prazo estabelecido para a entrega foi de 3 meses.

O desenvolvimento iniciou-se, novamente, com a assinatura do contrato e a equipe foi constituída pelos seguintes integrantes: 1 analista de sistemas/coordenador, 1 analista projetista, 1 programador HTML/JavaScript, 1 programador Java/C, 1 programador JSP e 2 testadores.

A partir daí, a equipe passou a se guiar pelo novo processo, onde o analista coordenador iniciou as atividades, preparando uma reunião para a coleta dos requisitos funcionais e não funcionais.

Ao final do desenvolvimento, o mais notável sinal de melhoria foi a entrega do módulo, que obedeceu aos prazos estabelecidos com o cliente.

A seguir, uma análise foi feita das atividades reatadas durante todo o processo, possibilitando ver as diferenças entre a metodologia utilizada pela empresa anteriormente e a metodologia proposto.

4.8.4 Análise dos dados

Inicialmente, as atividades da planilha formulário foram classificadas em: análise, projeto, implementação e testes.

De acordo com a classificação feita, fez-se um somatório dos tempos gastos com as atividades de cada fase. Podemos observar na figura 4.7 as diferenças de tempo com relação ao processo utilizado anteriormente.

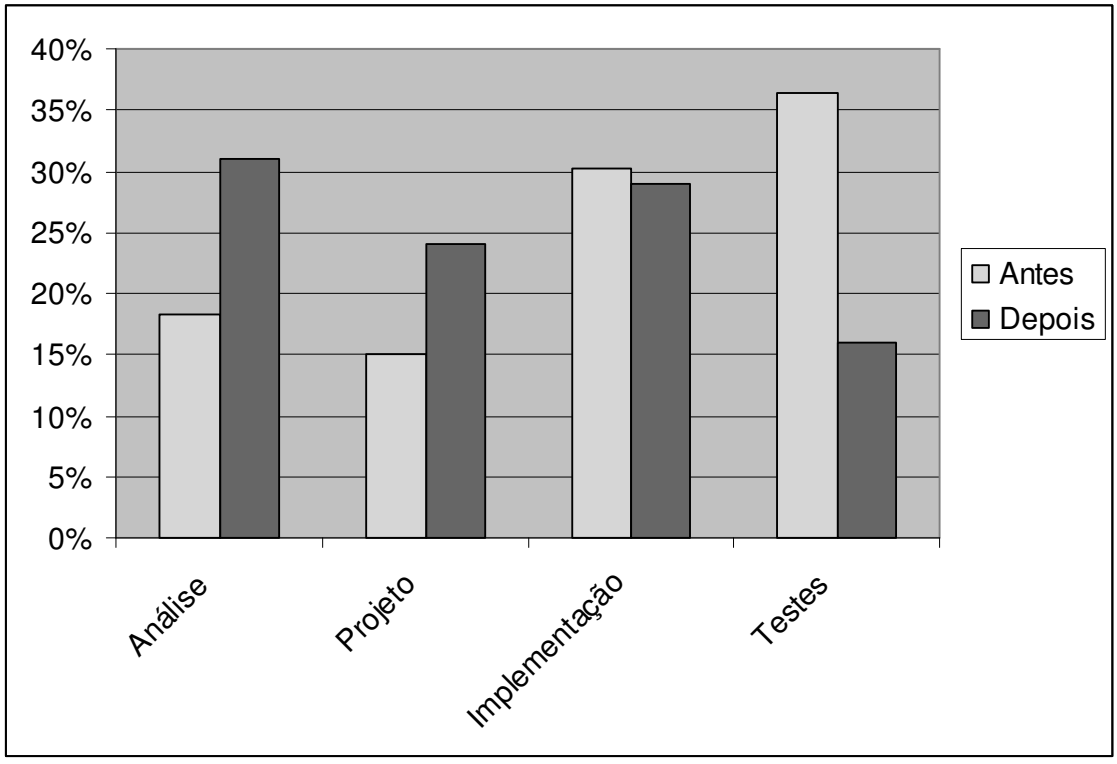

Figura 4.7: Gráfico comparativo entre os tempos gastos em cada fase, antes e depois.

Podemos observar que, com exceção da fase de implementação, o padrão de utilização dos tempos em cada fase do desenvolvimento mudou. As fase de Análise e Projeto consumiram muito mais tempo na nova metodologia do que anteriormente e a fase de Testes teve uma grande redução no gasto de tempo. Isto demonstra uma adequação aos padrões de consumo de tempo definido nas divesas metodologias.

Um ponto levantado na análise do processo utilizado anteriormente foi: que tipo de atividades a fase de implementação estava realizando, tendo em vista que ela, juntamente com a fase de teste, consumia cerca de 67% do tempo gasto no processo. A resposta foi que o maioria do tempo era gasto com correções e melhorias do software produzido.

Assim, também agora foram levantadas as atividades executadas pela fase de implementação, procurando classificá-las segundo os mesmos critérios definidos anteriormente:

- Construção de um novo módulo – construção com base na ata da reunião realizada entre o analista e o cliente.
- Correção de erros – correção de erros com base na lista de erros encontrados pelo analista nos testes.
- Implementação de melhorias – implementação de melhorias com base na ata da reunião incrementada pelo analista após os testes.

O resultado pode ser visto na figura 4.8, que representa os tempos gastos com cada atividade, trazendo a informação de que as implementações de novos módulos foram as principais consumidoras de tempo, com 82% do tempo total da fase de implementação. Pode-se concluir aqui que os módulos construídos tinham poucos erros e também poucas melhorias foram solicitadas pelo cliente.

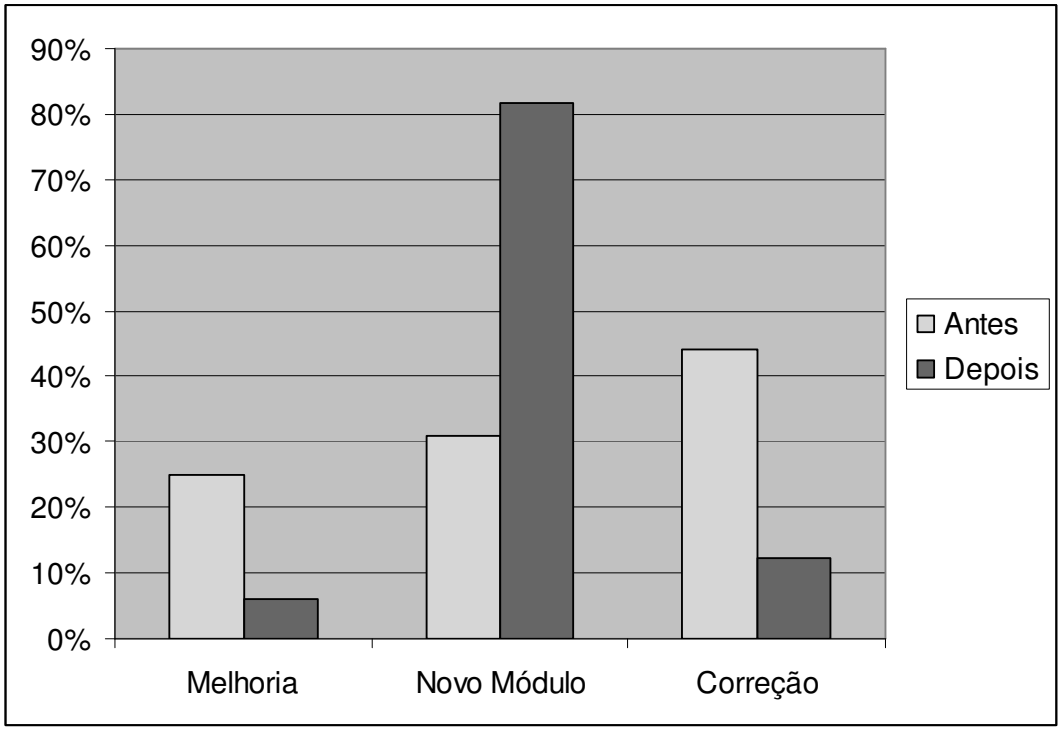

Figura 4.8: Gráfico comparativo dos tempos gastos com atividades de implementação.

4.8 Considerações Finais

O processo de desenvolvimento da WOPM Informática é bastante falho. Isto se deve, principalmente, ao fato de que o setor não sofreu uma organização inicial, e a equipe de desenvolvimento foi crescendo de acordo com a demanda.

Assim, de forma intuitiva, um processo foi estabelecido. Este processo dá ênfase à implementação, gastando pouco tempo com a produção de documentos de análise, o que é fatal para que o software produzido não atenda totalmente as expectativas do cliente, e no caso exposto acima, este fato se traduziu na forma de inúmeras novas implementações no software produzido, classificadas de melhorias, para atender aos anseios do cliente.

Isto deixa claro que é preciso uma grande mudança no processo produtivo, partindo da adoção de uma metodologia que seja acessível, de fácil implantação e que atenda principalmente ao processo produtivo. Em seguida faz-se necessário um detalhamento de cada atividade da metodologia escolhida, buscando estabelecer, de forma clara e objetiva, cada etapa do processo, possibilitando assim uma aplicação prática, onde poderemos colher os dados necessários para uma nova avaliação.

5 CONCLUSÕES E RECOMENDAÇÕES

5.1 Conclusões da Aplicação Prática da Metodologia

O estágio atual, em que se encontra o mercado globalizado, exige que a classe empresarial busque novas fórmulas alternativas de competitividade para poder continuar a desenvolver suas atividades, cujas alternativas devem ser rápidas e eficazes. Uma vez identificadas as necessidades dos clientes e imediatamente levadas aos processos de desenvolvimento de software da empresa em busca de oportunidades de melhoria, é o ponto fundamental neste contexto.

Neste trabalho procurou-se mostrar a importância e a utilidade da aplicação de métodos de desenvolvimento de software e, ao mesmo tempo, demonstrar que a "metodologia simplificada de desenvolvimento de software proposta" é uma ferramenta eficaz e que mapeia corretamente os requisitos do software e proporcionando assim a melhoria da qualidade do software produzido.

Portanto, se os empresários de pequeno e médio porte quiserem implantar uma metodologia de desenvolvimento de software que traga mais confiabilidade e qualidade ao software produzido, e que isso não implique em gastos exorbitantes com treinamentos e consultorias, eles têm na metodologia proposta uma ferramenta que, aliada a uma administração eficiente do projeto, proporcionar-lhes-á mais competitividade, deixando maiores resultados positivos no final do projeto. Por outro lado, reduzindo custos e repassando essa redução nos valores cobrados dos seus clientes, ficando estes mais satisfeitos, firmando cada vez mais usa fidelização.

Por sua vez, o monitoramento das atividades permite manter e melhorar essa metodologia, através de uma contínua avaliação de suas atividades, sob os aspectos de tempo, necessidade e utilidade de novos artefatos. No trabalho, o conceito de monitoração e medidas de tempo foi utilizado para aperfeiçoar a metodologia utilizada pela empresa, tomando por base as metas do projeto a serem atingida.

Na avaliação dos tempos, ficou evidenciado que existem falhas no processo de análise que precisam ser corrigidas, tanto na extração dos requisitos do sistema junto ao cliente quanto na produção de artefatos que traduzam de forma clara estes requisitos. Os desvios de tempo e custos são altos em todas as fases, demonstrando o desperdício de tempo e de dinheiro na elaboração do sistema, necessitando de uma metodologia eficaz que fizesse a correção destas distorções. Conhecidos esses desvios, buscou-se identificar as possíveis causas que travam o andamento dos processos como também detectou-se onde introduzir melhorias para que se atinja a meta ideal determinada pelas metas estabelecidas com o cliente.

5.2 Conclusões da Dissertação

Quanto a hipótese e ao problema de pesquisa, ficou demonstrado que é possível as empresas implantarem uma metodologia de desenvolvimento de software que seja simplificado e que se adapte ao porte da empresa, podendo-se, através da monitoração de seus processos estabelecer uma constante melhoria na metodologia. Em vista disso, o problema mencionado na Introdução deste trabalho foi reduzido sensivelmente.

O objetivo geral foi alcançado, pois a metodologia de desenvolvimento de software simplificada proposta, mostrou-se eficiente para este fim. Quanto aos objetivos específicos, também obteve-se êxito, uma vez que foi identificado na fundamentação teórica, que o método de desenvolvimento de software Fusion é o que melhor se adapta ao modelo de fábrica de software da empresa estudada. O método Fusion foi analisado e as atividades necessárias às fases de desenvolvimento foram conceitualmente detalhadas. A partir do detalhamento do Fusion, foi proposto uma metodologia simplificada, com as atividades de análise detalhadas, utilizando a notação UML, com o intuito de fornecer às empresas desenvolvedoras de software de pequeno e médio porte, uma alternativa que melhor adapta-se às suas necessidades e tamanho. Foi feito um levantamento na empresa W.O.P.M. Informática, buscando identificar a fase crítica do processo de desenvolvimento de software da empresa. O resultado deste levantamento

identificou a fase de análise como sendo a fase crítica. Foram realizados treinamentos sobre a metodologia simplificada para os membros da equipe, dando ênfase aos envolvidos com a fase de análise. Fez-se um novo levantamento durante um novo projeto e os resultados analisados. A metodologia foi implantanda na empresa de forma satisfatória, trazendo benefícios imediatos, como a diminuição do número de erros, diminuição do número de pedidos de melhorias, o cumprimento dos prazos estabelecidos com o cliente, a percepção de uma maior qualidade do software produzido e, ainda, uma redução no custo do produto.

Assim concluiu-se que o modelo pode ser implantado na forma de um modelo de fábrica de software e que a aplicação da metodologia pode trazer melhorias no processo de produção de software e um aumento da qualidade do software produzido.

Em face dessas análises, pode-se concluir que as empresas desenvolvedoras de software precisam estar continuamente em mudança para se adaptarem aos novos cenários advindos da crescente complexidade dos software a serem construídos. O enfrentamento com as concorrentes é cada vez mais acirrado, o que faz com que as mudanças devam ser rápidas para acompanhar o mercado.

Essas mudanças só serão possíveis, se a organização conhecer detalhadamente o seu método de desenvolver software para que possa buscar a melhoria contínua nos seus processos. Isso levará as organizações a se manterem competitivas e duradouras.

5.3 Sugestões para Trabalhos Futuros

A aplicação da metodologia proposta restringiu-se à fase de análise. Para que as empresas de desenvolvimento de software tenham um melhor aproveitamento dos seus recursos, faz-se necessário o detalhamento da fase de Projeto.

Devido a limitações desse trabalho, as lacunas deverão ser aproveitadas para novos trabalhos, podendo trazer contribuições aos interessados. A aplicação da

metodologia em empresas de um porte maior poderá evidenciar melhorias para tornar esta metodologia aplicável também a grandes projetos, como também, a utilização de outras medidas durante o monitoramento das atividades poderá complementar a metodologia proposta. Também a busca pela definição de um modelo de gestão de projeto para a metodologia proposta é uma sugestão para estudos posteriores.

REFERÊNCIAS BIBLIOGRÁFICAS

(ISO, 1995a) NBR ISSO / IEC 12207:1995. Tecnologia da informação - processos de ciclo de vida de software, 1997.

(ISO, 1995b) ISO / IEC 12207. Information technology - software life cycle processes. International Standard Organization, 1995.

A Survey of Structured and Object-Oriented Software Specification Methods and Techniques, Roel Wieringa, ACM Computing Survey, dez/98.

A Survey of Structured and Object-Oriented Software Specification Methods and Techniques, Roel Wieringa, ACM Computing Survey, dez/98.

An Integrated Approach to Software Engineering, Pankaj Jalote, 1997.

ALBUQUERQUE, Fernando. Fernando. Disponível através do protocolo http em: http://www.cic.unb.br/docentes/fernando/fernando.html. (Out, 1999).

BECK, Kent. Extreme Programming Explained: Embrace change. Reading, Massachusetts: Ed. Addison-Wesley, 2000.

BERARD, E. V. Berard. Essays on Object-Oriented SoftwareEngineering. Addison-Wesley, 1993.

BIRTWHISTLE, G.M. Birtwhistle, O.J. Dahl, B. Myhrhaug, K. Nygaard.Simula Begin. Chartwell-Bratt, 1979.

BMS. Belgo Mineira Sistemas - Fábrica de Software. Disponível em: <http://www.bms.com.br/content/interna_fabrica.asp >. Acesso em 31/07/2002.

BOEHM, Barry. A Spiral Model of Development and Enhacement. IEEE Computer, vol 21, nº 5, 1988, pp.61-82.

BONA, Cirstina. - Avaliação De Processos De Software: Um Estudo de Caso em Xp e Iconix. UFSC, Florianópolis, 2003.

BOOCH, G. - Object-Oriented Design with Applications, Benjamin Cummings, Redwood City, CA, 1991.

BOOCH, Grady Booch. Object-Oriented Analysis and Designwith Applications, 2ª edição. Addison Wesley, 1994.

BOOCH, Grady Booch. Object-Oriented Development. IEEETrans. Software Engineering, vol. 12, no. 2, Fevereiro1986

BROOKS, F. P. The Mythical Man Month: Essays in Software Engineering. Yourdon Press, Englewood Cliffs, New Jersey, 1982.

BROOKS, Frederick P. Brooks Jr.. No Silver Bullet - Essenceand Accidents of Software Engineering. InformationProcessing '86. Elsevier Science Publishers B.V.,1986

COAD, Peter Coad, Edward Yourdon. Object-OrientedAnalysis, 2ª edição. Yourdon Press, 1991.

COAD, Peter Coad, Eric Lefebvre, Jeff de Luca. JavaModeling In Color With UML: Enterprise Componentsand Process. Prentice Hall, 2000.

COLEMAN, D. et al - Object-Oriented Development: THE FUSION METHOD, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1994.

COLEMAN, Derek et al. Desenvolvimento orientado a objetos: o método fusion. Rio de Janeiro: Campus, 1996.

COTTON, Todd. Evolutionary Fusion: A Customer Oriented Incremental Life Cycle for Fusion. Disponível através do protocolo http em: http://www.hp.com/hpj/aug96/augart3.htm (Nov, 1999).

DEMARCO, Tom DeMarco. Structured Analysis and SystemSpecification, Yourdon Press, 1978.534 CENTRO ATLÂNTICO – COLECÇÃO TECNOLOGIAS – UML, METODOLOGIAS E FERRAMENTAS CASE.

DIJKSTRA, B. W. Dijkstra. Programming Considered as a HumanActivity. Proceedings of the 1665 IFIP Congress,North Holland Publishing Company, 1965.

DIJKSTRA, B. W. Dijkstra. Structured Programming. NATOConference on Software Engineering, Outubro 1969.

Engenharia de Software, Roger S. Pressman, Makron Books, 1995.

EXTEND. Extend Software - Fábrica. Conceito, Vantagens e Etapas. Disponível em: <http://www.extend.com.br/fabrica.asp>. Acesso em 31/07/2002.

FIRESMITH, Donald Firesmith. Object-Oriented RequirementsAnalysis and Logical Design: A Software EngineeringApproach. John Wiley & Sons, Inc., 1996.

FOWLER, Martin Fowler. The New Methodology. SoftwareDevelopment Magazine, Dezembro 2000.

FRANZEN, Pablo Leandro. Fase de Definição de Requisitos para o Método Fusion. Caxias do Sul, UCS, 1999.

FUGGETTA, A. Software process: a readmap. In: The Future of Software Engineering, A. Finkelstein (ed), 2000.

GANE, Trish Gane, Chris Sarson. Structured SystemsAnalysis. McDonnell Douglas, 1982.

HAREL, D. - Statecharts: a visual formalism for complex systems. Science of Computer Programming, 8 , p. 231-74, 1987.

JALOTE, Pankai. An Integrated Approach to Software Engineering, 1997.

JEFFRIES, Ron. XP Magazine Contents: What is Extreme Programming?. XProgramming.com – an Extreme Programming Resource, nov, 2001. Disponível em: <http://www.xprogramming.com/xpmag/index.htm>. Acesso em: 21 mai. 2003.

KRUTCHEN, Philippe. The Rational Unified Process. 2ª Edição. Addison Wesley, 2000.

LEVESON, N. G. Leveson, J. L. Stolzy. Safety Analysis UsingPetri Nets. IEEE Trans. on Software Engineering,Vol. 13, No. 3, Março 1987.

LINDVALL, M. e RUS, I. Process diversity in software development. IEEE Software, v.17, n. 4, jul./ago. 2000.

MACGRAW-HILL. Software Engineering: A Practitioner's Approach, McGraw-Hill, 2000.

MAFFEO, Bruno., Engenharia de Software e Especificações de Sistemas, 1ª Edição, Rio de Janeiro : Campus, 1992.

MARQUES, Sandro. Fábrica de Software: Requisitos. In: FENASOFT, 2001, São Paulo. Anais Eletrônicos. Disponível em: < http://www.fenasoft.com.br/congressista/artigos/fabricasoftware_smarques.doc>. Acesso em 31/07/2002.

MARTIN, J. Martin, J. J. Odell. Object-Oriented Analysis andDesign. Prentice Hall, 1992.

MARTIN, J. Martin. Information Engineering: Introduction Vol. 1.Prentice Hall, 1989.

PAGE-JONES, Meiler Page-Jones. The Practical Guide to StructuredSystems Design. Yourdon Press, 1980.

PRESSMAN, R. S. Software engineering - a practitioner's approach, 5ª Edição. Nova York, NY, McGraw-Hill, 2000.

PRESSMAN, Roger S. Engenharia de Software. Makron Books, 1995.

Real-Time UML Second Edition, Bruce P. Douglas, Addison-Wesley, 2000.

REED, K. Software Engineering - a new millenium ? IEEE Software, jul-ago 2000.

REZENDE, Desnis Alcides. Engenharia de software e sistemas de informações, Rio de Janeiro : Brasport, 1999.

ROCHA, Ana Regina Cavalcanti da., José Carlos Maldonaldo e Kival Chaves Weber. Qualidade de Software -Teoria e Prática, São Paulo : Prentice Hall, 2001.

RUMBAUGH, J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W.Lorenzen. Object-Oriented Modeling and Design.Prentice Hall, 1991.

SCHACH, Stephen Schach. Classical and Object-OrientedSoftware Engineering with UML and Java, 4ª edição.McGraw-Hill, 1999.

SEBRAE: A micro e pequena empresa no Brasil, 2003. Disponível em: <http://www.sebrae.com.br>. Acesso em: 01 jun. 2003.

SHLAER, S. Shlaer, S. J. Mellor. Object-Oriented SystemsAnalysis: Modeling the World in Data. Yourdon Press,1988.

SILVA, Alberto M. R.; VIDEIRA, Carlos A. E. UML, Metodologias e Ferramentas Case. Lisboa: Centro Atlântico, 2001.

Software Engineering: A Practitioner's Approach, McGraw-Hill, 2000.

SPIVEY, J. M. Spivey. Understanding Z: A SpecificationLanguage and its Formal Semantics. CambridgeUniversity Press, 1988.

STROUSTRUP, B. Stroustrup. What is Object-Oriented Programming.IEEE Software, vol. 5, no. 3, pp 10-20, Maio 1988.

UML Essencial, 2 edição, M. Fowler e K. Scott, Bookman, 2000.

WEAVER, Philip Weaver, Nick Lambrou, Matthew Walkley.Practical SSADM Version 4+, 2ª edição. FinancialTimes Prentice Hall, 1998.

WELLS, Don. Extreme Programming: A gentle introduction, 2002. Disponível em: <http://www.extremeprogramming.org>. Acesso em: 20 mai. 2003.

WIRFS-BROCK, R.; Wilkerson, B.; Wiener, L.. - Designing Object-Oriented Software, Prentice Hall International, Englewood Cliffs, 1990.

WIRTH, Niklaus Wirth. Program Development by StepwiseRefinement. Communications of the ACM, Vol. 14,No. 4, Abril 1971.

YOURDON, Edward Yourdon. Modern Structured Analysis.Prentice Hall, 1999.

YOURDON, Edward, Administrando o Ciclo de Vida do Sistema, 1ª Edição, Rio de Janeiro : Campus, 1989.

APÊNDICE

APÊNDICE A - FORMULÁRIO DE ACOMPANHAMENTO DE ATIVIDADES

Formulário distribuído através de planilha eletrônica compartilhada com a equipe de desenvolvimento.

Atividade	Como foi feita	Ferramentas utilizadas	Artefatos de Entrada	Artefatos de Saída	Tempo gasto	Data	Quem