KARINA BORTOLUCCE

APLICAÇÃO DE UMA METODOLOGIA NA MANUTENÇÃO EVOLUTIVA DE UM SISTEMA DE SOFTWARE

Monografia a ser apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de MBA em Engenharia de Software.

Área de Concentração:
Engenharia de Software

Orientadora:
Profª. Maria Alice G. V. Ferreira

São Paulo
2003

À minha família e meus amigos que me apoiaram com carinho e paciência em todos os momentos deste trabalho.

AGRADECIMENTOS

À professora Maria Alice pela orientação, dedicação e incentivo no desenvolvimento desta Dissertação.

Ao colega Marcelo Gomes que contribui com seu conhecimento a acrescentar na pesquisa.

Aos meus pais Nelson e Sônia, por estarem sempre me apoiando.

As minhas irmãs Elaine e Fernanda e aos meus amigos Denis e Fernando por vossa paciência e ajuda.

A todos que colaboraram, direta ou indiretamente, para a execução deste trabalho.

RESUMO

Este trabalho apresenta um estudo de metodologias, processo de manutenção e engenharia reversa e a situação atual da empresa em que trabalha a autora com o objetivo de fornecer uma metodologia para melhorar o processo de manutenção de software vigente. A proposta envolve o uso de engenharia reversa para a geração dos modelos da Análise Essencial para um sistema já implantado, que é o centro dos processos de negócio da empresa, que são muito importantes para a sua atualização, de acordo com a manutenção. Propõe-se também um processo de manutenção com fases bem definidas visando à qualidade.

ABSTRACT

This paper presents a study of methodologies, maintenance process and reverse engineering and the current situation of the company, where the author works with the purpose to provide a methodology in order to improve the current software maintenance process. The proposal involves the use of reverse engineering in the generation of the models of Essential Analysis for an implanted system that is the core of company business processes, which are very important in its update, in accordance with the maintenance. A process of maintenance, which has gotten quite definite phases, is also proposed aiming at quality.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CMM – Capability Maturity Model

DED – Diagrama de Estrutura de Dados

DFD – Diagrama de Fluxo de Dados

DOC – Documento de Ordem de Crédito

IEEE – Institute of Electrical and Electronics Engineers

ISO – International Organization for Standardization

KPAs – Key Process Areas

MDS – Metodologias de Desenvolvimento de Sistemas

OO – Orientação a Objetos

RUP – Rational Unified Process

SEI – Software Engineering Institute

SEPG – Software Engineering Process Group

Std – Standard

UML – Unified Modeling Language

XP – Extremme Programming

Capítulo 1 – INTRODUÇÃO

Às vésperas da virada do século, com o cenário de transformações constantes do mercado mundial, a temática prioritária no campo empresarial passou a ser a competitividade. A necessidade de se impor em um mercado globalizado, sem fronteiras, faz com que empresas vivam um processo contínuo de transformação. Diante desse contexto, os grandes desafios estão na busca da garantia da qualidade e produtividade. No âmbito da tecnologia da informação essa realidade não é diferente.

A qualidade de software é um dos assuntos mais atuais em discussão na comunidade de Engenharia de Software. Embora os primeiros esforços no sentido de se produzir software com maior qualidade e produtividade datem da década de 70, foi na segunda metade da década de 90 que uma série de novos conceitos e abordagens alcançaram maturidade e visibilidade. Com os objetivos relacionados, para a elevação do nível de qualidade do *software* é recomendável utilizar como referências, modelos reconhecidos internacionalmente, como as normas ISO e o modelo CMM. Dentre estes, o SEI-CMM é o mais conhecido e utilizado.

A conformidade com esses modelos, no entanto, exige um grande esforço na definição de procedimentos e padrões, e envolve a implantação de uma metodologia de desenvolvimento que descreva as atividades, produtos e responsáveis pelas diversas etapas do ciclo de vida do *software*. Neste sentido, metodologias orientadas a objetos, que estimulem um desenvolvimento iterativo e incremental, com foco em análise de riscos, têm se mostrado convenientes e úteis.

O *Rational Unified Process* – RUP – apresenta características adequadas aos sistemas de qualidade devido a sua ampla abrangência, com a definição de atividades que contemplam desde o planejamento do projeto, até os processos de teste e gerência da configuração do *software*. Mas, apesar do RUP estar sendo adotado como metodologia padrão por várias organizações em todo o mundo, percebe-se um relevante grau de incerteza no que diz respeito a seu nível de conformidade com modelos internacionais de qualidade.

Assim sendo, um dos maiores desafios das organizações de software hoje é, justamente, aplicar em seus processos de desenvolvimento tanto os novos conceitos de engenharia de software quanto as práticas de qualidade preconizadas pelo CMM e outros modelos.

1.1 O contexto

Nos últimos anos, as organizações de desenvolvimento de software têm aumentado sua percepção em relação aos problemas que tipicamente as tocam. Software com *bugs*, prazos e orçamentos não cumpridos e insatisfação de clientes e usuários são eventos muito mais freqüentes do que se desejaria.

Já desde a década de 70 existe um consenso na comunidade de Engenharia de Software de que estes problemas estão, em grande parte, relacionados ao fato de que o desenvolvimento de software é muitas vezes realizado de forma "artesanal", isto é, através de métodos improvisados pelos desenvolvedores, os quais, por sua vez, muitas vezes dependem muito mais de seu talento individual – nem sempre abundante – do que de uma sólida formação, acompanhada de métodos formais que dirijam suas atividades. Isto levou ao desenvolvimento e introdução das chamadas Metodologias de Desenvolvimento de Sistemas (MDS), através das quais se procurava padronizar boas práticas de Engenharia de Software, normalmente vinculadas a técnicas específicas, tais como a Análise Estruturada de Sistemas ou a Modelagem de Dados.

O sucesso na implantação e uso das MDS foi relativo. Nas (poucas) organizações onde existiam as condições para seu sucesso, resultados significativos foram alcançados. Em muitos lugares, no entanto, tais resultados não foram os esperados. Dentre os motivos que levaram ao relativo insucesso das MDS, um dos mais importantes é o foco excessivo que estas colocam nas atividades de Engenharia, em detrimento das atividades de Gerenciamento. Boas técnicas de desenvolvimento não adiantam se o projeto como um todo é mal conduzido.

1.1.1 CMM

Combinando esta percepção com os conceitos mais genéricos do Gerenciamento da Qualidade Total, a comunidade começou a priorizar o foco das iniciativas de melhoria na definição de melhores processos de gerenciamento de projetos de software. A consolidação destas idéias se deu através do modelo SEI-CMM, da Carnegie-Mellon University.

O CMM procura orientar a organização no sentido de implementar a melhoria contínua do processo de software, e o faz através de um modelo de cinco níveis, priorizando de forma lógica as ações a serem realizadas. Quanto maior o nível, maior a maturidade da organização, o que se traduz em maior qualidade do produto final, prazos e custos mais baixos e maior previsibilidade em cronogramas e orçamentos.

No nível 1, chamado de Inicial, o desenvolvimento é caótico. Não existem procedimentos padronizados, estimativas de custos e planos de projeto. Cada qual desenvolve como quer, não existe documentação e não há mecanismos de controle que permitam ao gerente saber o que está acontecendo, identificar problemas e riscos e agir de acordo com as exigências de cada situação. Como conseqüência, os desvios não são corrigidos e ocorrem os problemas como prazos não cumpridos, orçamentos estourados, software sem qualidade e usuários insatisfeitos. Na verdade, raramente existe um cronograma ou um orçamento. Infelizmente, estima-se que mais de três quartos das empresas norte-americanas encontram-se neste nível, e não há razões para acreditar que a situação seja melhor no Brasil.

Para passar ao nível 2, a organização deve instituir controles básicos de projeto, incluindo o Gerenciamento de Requisitos e de Projetos – técnicas para planejar e estimar o esforço em projetos, e controlar o progresso –, Controle Gerencial – verificação pela Gerência do progresso do projeto em momentos pré-determinados, incluindo a qualidade dos produtos –, a instituição de um Grupo de Garantia de Qualidade e de procedimentos básicos de Gerenciamento de Configuração – para garantir que mudanças no projeto e manutenções solicitadas não destruam o que já foi feito, garantindo um mínimo de estabilidade no desenvolvimento; nada é mais

deletério para um projeto do que requisitos que mudam constantemente e sem controle.

Chegando ao nível 2, chamado Repetível, a organização está em condições de ter maior controle sobre seus projetos, e pode-se esperar que as estimativas sejam mais precisas, já que se desenvolve uma base histórica intuitiva, e a qualidade do software produzido seja maior. No entanto, caso a empresa enfrente o desafio de atacar projetos de características distintas daquelas a que está acostumada, usando uma nova tecnologia, por exemplo, esta informação será irrelevante, e a empresa poderá regredir ao nível 1.

Para passar ao nível 3, Definido, é necessário introduzir uma Metodologia de Desenvolvimento formal padronizada, com um ciclo de vida definido, acompanhada de métodos, técnicas e ferramentas apropriadas, como inspeções e técnicas abrangentes de teste. É o momento também de estabelecer o SEPG, isto é, o time encarregado exclusivamente da melhoria contínua do processo de software. Ao chegar a este nível, a empresa terá um fundamento claro para desenvolver sistemas e também para melhorar o próprio processo, especialmente quando surgirem crises.

No nível 3, entretanto, os controles ainda são basicamente qualitativos, não havendo meios de quantificar a qualidade dos produtos e a eficiência do processo. Assim, a empresa deve estabelecer métricas de forma a medir características específicas dos produtos. A forma de coletar, armazenar e analisar estes dados é definida e, com base nesta informação, pode-se sugerir melhorias específicas nos produtos. Neste ponto, a empresa estará no nível 4, ou Gerenciado.

Para subir do nível Gerenciado para o último nível, o de Otimização, deve-se estabelecer meios para a coleta automática de métricas e para a utilização da informação coletada de forma a prevenir problemas. A idéia é analisar as causas dos problemas e atacá-las para evitar que voltem a ocorrer. Enquanto os dados coletados no nível 4 podem informar, por exemplo, quantos erros existem em um programa, a preocupação no nível 5 é melhorar o processo para evitar que tais erros aconteçam no próximo projeto.

1.1.2 RUP e OO

Por outro lado, olhando sob o aspecto das técnicas de desenvolvimento, o desenvolvimento orientado a objetos (OO) sofreu um forte processo de amadurecimento. As técnicas OO foram desenvolvidas para superar uma série de limitações das técnicas estruturadas, então hegemônicas nas organizações de desenvolvimento. Através de conceitos como reutilização de código, encapsulamento, herança e polimorfismo, o desenvolvimento OO promete maior qualidade e produtividade no desenvolvimento, dadas condições iguais de gerenciamento de projetos.

Uma das maiores dificuldades encontradas pela Orientação a Objetos em suas tentativas de obter maior visibilidade e uso foi a falta de padronização, especialmente no que diz respeito à nomenclatura e notação. Modelos OO "pululavam", oriundos das mais diversas origens, gerando muitas vezes confusão entre os desenvolvedores que iniciavam seu estudo. Some-se a isso a dificuldade, decorrente da falta de padrão, em se disponibilizarem ferramentas CASE para modelagem OO e ter-se-á explicação para parte da dificuldade que as técnicas OO tiveram, apesar de seus enormes méritos, em alcançar maior divulgação e adoção por parte das organizações.

Neste sentido, um avanço muito significativo foi o surgimento da UML – *Unified Modeling Language* –, definindo um padrão em termos de notação, e facilitando, portanto, o estudo e a adoção da OO pelas organizações, na medida em que esta padronização permitiu a criação de ferramentas adequadas e a simplificação do treinamento dos desenvolvedores.

No entanto, a UML restringia-se a definir um padrão de notação, sem prescrever um processo de desenvolvimento de software. Em outras palavras, a organização que quisesse desenvolver software segundo o paradigma OO já dispunha de uma notação padronizada e universalmente aceita, mas tinha a necessidade de adicionar à UML um processo OO de desenvolvimento por conta própria. O RUP – *Rational Unified Process* – foi desenvolvido para suprir esta lacuna, sugerindo às organizações um processo de desenvolvimento OO coerente com a UML e, através de customização, aplicável a uma grande diversidade de projetos de desenvolvimento.

A estrutura do RUP permite, inclusive, que ele seja customizado para a definição de processos de desenvolvimento baseados em técnicas não-OO, como a Análise Essencial. Isto é muito útil, na medida em que organizações com grande número de sistemas legados, desenvolvidos com uso de técnicas estruturadas, possam adotar o RUP tanto para a manutenção de sistemas estruturados legados – e posterior migração para OO – quanto para o desenvolvimento de novos sistemas totalmente dentro dos conceitos OO. Isto permite economia de expressão e significativas economias em treinamento.

1.1.3 CMM e Metodologias de Desenvolvimento de Sistemas (MDS)

Outro aspecto interessante do RUP é que ele inclui, em sua estrutura, processos gerenciais, conforme exigido pelo CMM no Nível 2, reconhecendo explicitamente a necessidade de tais processos para garantir o desenvolvimento ordenado de software.

Embora o CMM priorize a melhoria do processo de software, colocando o foco em primeiro lugar nas práticas de gerenciamento de projetos (Nível 2), isto não significa que as técnicas de Engenharia de Software devem ser desprezadas em um primeiro momento. De fato, o nível 3 do CMM é o contexto por excelência da melhoria das práticas de engenharia, quando então a organização deve selecionar as melhores práticas de engenharia de seus diversos projetos e definir um Processo Padrão de Desenvolvimento (MDS), a ser utilizado, de forma customizada, em cada projeto. Este fato, porém, não deve levar à ilusão de que não é necessário usar uma MDS no nível 2. Neste nível, o CMM apenas não exige que a MDS seja padronizada, sendo que cada projeto pode selecionar a sua MDS.

Na prática, a experiência mostra que é útil a definição de uma MDS ainda durante o processo de implantação do Nível 2 do CMM na organização, particularmente em situações onde exista grande volume de manutenção e forte integração entre sistemas. A possibilidade, prevista no Nível 2, de cada projeto definir sua própria MDS é inviável, quando a maioria dos projetos é de manutenção, de curto prazo e abrangendo software que, embora desenvolvidos em diferentes projetos, apresentam forte integração.

Assim sendo, é recomendável que as organizações que estejam implantando, ou que estejam considerando implantar, o Nível 2 do CMM também incluam no projeto de melhoria a elaboração de uma MDS padronizada a ser usada por todos os projetos. Neste sentido, aquelas organizações que tenham também por objetivo a implantação de novas técnicas de desenvolvimento podem se beneficiar da sinergia existente entre o CMM e o RUP.

É importante ressaltar que o RUP tem como pré-requisito que a organização que o implanta já possua um mínimo de organização em seu processo de desenvolvimento. Em outras palavras, dificilmente uma organização se beneficiará das vantagens trazidas pelo RUP, se não colocar em prática, ao mesmo tempo, processos disciplinados como os preconizados pelo CMM ou equivalentes.

1.2 Motivação

A partir do momento em que um software é colocado em uso, novos requisitos emergem, e os requisitos existentes são modificados à medida que a empresa que utiliza esse software, passa por modificações. Partes do software podem precisar de modificações para corrigir os erros encontrados na operação, para melhorar seu desempenho ou por exigência de outras características não funcionais. Tudo isso indica que, depois de serem entregues, os sistemas de software sempre evoluem, em resposta às exigências de mudanças.

A empresa em que trabalha a autora da monografia, encontra-se na situação de um sistema já implantado que sofre freqüentes manutenções evolutivas e corretivas, onde a taxa destas modificações, é regida pelo processo de tomada de decisões por parte da organização.

Para esse trabalho de desenvolvimento e manutenção, nota-se a dificuldade encontrada pelos profissionais quando se responsabilizam pela manutenção de produtos que não são de seu total conhecimento. Por não se possuir documentação do sistema, o processo de manutenção torna-se lento e as contínuas consultas a colegas de trabalho que já estiveram envolvidos anteriormente com este software – única forma de obtenção de conhecimento sobre ele – podem se tornar incômodas a eles,

diminuindo a sua produtividade quanto a seus próprios afazeres e desviando sua atenção de seu trabalho prioritário. A falta do cumprimento das fases de Análise, Especificação, Desenvolvimento, Testes e Implementação e inexistência da documentação necessária, implica na dificuldade de compreender a integração do sistema como um todo e de se apresentar, de maneira mais formal, ao cliente.

O objetivo desta monografia, é aplicar uma metodologia e um processo adequados a este problema, prevendo os benefícios advindos de se possuir uma documentação formal.

É importante ressaltar que o trabalho citado tem também a finalidade de manter um alto nível de qualidade no processo de desenvolvimento de software bem como do produto final.

1.3 Perspectiva de Contribuição

Pretende-se com este trabalho:

- Levantar os benefícios da aplicação de uma metodologia na manutenção de um sistema já em operação, mas que vem sofrendo contínuas alterações;
- Estabelecer o processo de aplicação de uma metodologia adequada ao sistema já implantado, através do estudo da situação atual, do levantamento das características da ferramenta utilizada na empresa – em torno da qual se dá toda a codificação efetuada - e do mínimo de documentação que o sistema possui;
- Melhorar o trabalho das pessoas envolvidas na manutenção evolutiva de um sistema de software, da empresa da autora.

1.4 Metodologia

A metodologia empregada consistiu dos seguintes passos:

- Estudo de metodologias da Engenharia de Software, principalmente as mais recentes – orientadas a objeto – como RUP – *Rational Unified Process* – e XP – *Extremme Progrmming*. Estudo sobre qualidade de software, que

envolveram CMM e ISO. Este estudo aponta que a qualidade de um produto de software está mais relacionada ao processo de desenvolvimento que o gerou do que com o software propriamente dito;

- Levantamento do processo atualmente adotado pela empresa em questão. A partir deste levantamento deu-se início à proposta da metodologia a ser adotada, visando à melhoria de qualidade de software;
- Sob o prisma da qualidade, Pressman (2002, p. 19) ressalta que "o processo de Engenharia de Software define a estrutura para um conjunto de áreas–chave de processo (KPAs – *key process areas*), que deve ser estabelecido para a efetiva utilização da tecnologia de engenharia de software. As áreas–chave de processo formam a base para o controle gerencial de processos de software e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários etc) são produzidos, marcos são estabelecidos, qualidade é assegurada e modificações são adequadamente geridas". Assim, a pesquisa de um processo de Engenharia de Software inclui duas outras variáveis a serem estudadas: os métodos técnicos e as ferramentas que fornecem o apoio automatizado a eles. No caso do software em questão, existe uma ferramenta envolvida, a qual é utilizada na geração do produto final a ser encaminhado para operação. O estudo desta ferramenta é o passo seguinte neste processo de levantamento;
- Uma proposta para a aplicação de uma metodologia estruturada para a empresa da autora sob as metodologias estudadas.

1.5 Estrutura de Trabalho

Esta monografia está estruturada como se descreve a seguir.

No Capítulo 1 foi feita uma rápida explanação sobre a situação atual de desenvolvimento de software, bem como sobre critérios de qualidade que norteiam este desenvolvimento. Foram apresentados também os objetivos gerais desta pesquisa e descreveu-se o contexto em que ela se insere.

No Capítulo 2 constam metodologias pesquisadas, processo de manutenção e conceitos de engenharia reversa que fornecem o conteúdo para a geração da proposta deste trabalho.

No capítulo 3 é descrito o processo atual da empresa relatando toda documentação utilizada e as pessoas envolvidas no processo de manutenção.

No capítulo 4 apresenta-se a proposta da aplicação de uma metodologia gerada a partir do estudo efetuado nos capítulos anteriores, para obter-se um melhor processo de manutenção na empresa da autora.

No capítulo 5 são apresentadas as considerações finais e a conclusão do trabalho.

Capítulo 2 – METODOLOGIAS PESQUISADAS

Neste capítulo analisam-se várias metodologias, quanto ao processo de manutenção de software. Primeiramente, é feita uma compilação do processo de manutenção, segundo as visões de vários pesquisadores, tais como expressas em Yourdon (1990), Sommerville (2003) e Paula Filho (2001). Depois, estudam-se a *Extreme Programming* (XP) e o *Rational Unified Process* (RUP). Finalmente, comparam-se estes processos quanto a alguns aspectos, considerados relevantes a este estudo.

2.1 Manutenção de Software

A manutenção de software é o processo geral de modificação de um sistema depois que ele foi colocado em uso. As modificações podem ser simples, destinadas a corrigir erros de códigos, mais extensas, a fim de corrigir os erros de projeto, ou significativas, com a finalidade de corrigir erros de especificação ou acomodar novos requisitos, segundo Ian Sommerville (2003) e ANSI/IEEE (1983).

A partir do momento em que o software foi implantado, novos requisitos emergem, e os requisitos existentes são modificados à medida que a empresa que utiliza esse software, passa por modificações. Por isso, as mudanças nos produtos de software são de grande importância, pois estas organizações dependem inteiramente de seus sistemas de software.

A manutenção é, portanto, uma continuação do processo de desenvolvimento de sistema, com atividades associadas de especificação, projeto, implementação e testes.

A manutenção é facilitada pelos seguintes fatores de manutenibilidade:

- Disponibilidade de pessoal qualificado;
- Desenho adequado e bem documentado;
- Uso de linguagens padronizadas;
- Uso de ambientes padronizados de desenvolvimento e operação;
- Documentação padronizada;

- Disponibilidade de casos e procedimentos de teste;
- Ferramentas e procedimentos padronizados de gestão de configurações;
- Existência de um processo de manutenção definido.

De acordo com Ian Sommerville (2003), Lehman e Belay (1985) propuseram um conjunto de leis invariáveis e amplamente aplicáveis – Leis de Lehman – referentes a mudanças nos sistemas. Essas leis estão apresentadas na Tabela 2.1.

Tabela 2.1 – Leis de Lehman (baseada em Ian Sommerville (2003))

Lei	Descrição
Mudança contínua	Um programa utilizado em um ambiente do mundo real, necessariamente, tem de ser modificado ou se tornará de maneira progressiva menos útil nesse ambiente
Aumento da complexidade	À medida que um programa em evolução se modifica, sua estrutura tende a se tornar mais complexa. Recursos extras precisam ser dedicados a preservar e simplificar a estrutura.
Evolução de um programa de porte grande	A evolução do programa é um processo auto-regulador. Os atributos do sistema, como tamanho, tempo entre *releases* e número de erros relatados, são aproximadamente invariáveis para cada *release* do sistema.
Estabilidade Organizacional	Durante o tempo de duração de um programa, sua taxa de desenvolvimento é aproximadamente constante e independente dos recursos dedicados ao desenvolvimento do sistema.
Conservação da familiaridade	Durante o tempo de duração de um sistema, as mudanças incrementais em cada *release* são aproximadamente constantes.

A primeira lei coloca que a manutenção do sistema é um processo inevitável: surgem novos requisitos de acordo com as modificações que ocorrem no ambiente em que o software se insere, as quais devem ser implementadas; o sistema modificado é introduzido novamente no ambiente, promovendo mais mudanças, e dessa forma, o processo de evolução torna-se contínuo.

A segunda lei especifica a degradação da estrutura do sistema, na medida em que este é modificado. Para evitar que isso aconteça, é necessário investir na manutenção preventiva.

A terceira lei é sugerida por Lehman e Belady (1985) como resultado de fatores estruturais e organizacionais fundamentais, porque ela propõe que os sistemas de grande porte têm uma dinâmica própria, estabelecida em um estágio inicial do processo de desenvolvimento. Isso determina as tendências gerais do processo de manutenção do sistema e limita o número de possíveis mudanças no mesmo.

A quarta lei sugere que as mudanças nos recursos ou no pessoal envolvido têm efeitos imperceptíveis na evolução de longo prazo do sistema. Essa lei confirma também que grandes equipes de desenvolvimento são improdutivas quando atividades indiretas de comunicação dominam o trabalho da equipe.

A quinta lei de Lehman diz respeito aos estágios de alterações em cada *release* de sistema. Quanto mais funcionalidade for introduzida no sistema, maior poderá ser o número de reparos a serem incluídos no próximo *release*. Contudo a lei propõe que não se deve orçar grandes aumentos de funcionalidade em cada versão sem levar em consideração a necessidade de reparo de defeitos.

2.1.1 Tipos de Manutenção de Software

Existem quatro diferentes tipos de manutenção de software:

1. Manutenção corretiva: consiste na remoção dos defeitos remanescentes após o fim do projeto de desenvolvimento;
2. Manutenção adaptativa: manutenção necessária quando algum aspecto do ambiente de sistema é modificado exigindo a adaptação do produto a novos requisitos;
3. Manutenção evolutiva: melhorias solicitadas pelos usuários em resposta a mudanças organizacionais ou de negócios e, em outros casos, se refere a manter a funcionalidade do sistema, melhorando sua estrutura e seu desempenho.
4. Manutenção preventiva: manutenção que procura localizar os defeitos antes que estes se manifestem em operação.

Para Wilson P. de Pádua Filho (2001) "a manutenção adaptativa e perfectiva são apenas disfarces para formas indisciplinadas de desenvolvimento... Na maioria das

vezes, são requisitos listados para novas versões de um produto, impedindo que novos defeitos sejam introduzidos numa fração significativa de mudanças".

2.1.2 Custos de Manutenção de Software

Reparar defeitos em sistemas tal como, adaptá-lo a um novo ambiente ou novos requisitos, exige uma grande parte do esforço de manutenção. Por isso, a manutenção é representada como uma atividade de processo separada.

Ian Sommerville (2003), ressalta que os custos de manutenção do sistema representam uma grande proporção do orçamento da maioria das organizações que utilizam sistemas de software. Estes custos, variam de um domínio de aplicação para outro. Para os sistemas de aplicação de negócio, um estudo feito por Guimarães (1983) mostrou que os custos de manutenção eram amplamente comparáveis com os custos de desenvolvimento de sistemas.

De modo geral, é eficaz investir esforço no projeto e na implementação de um sistema para que posteriormente o custo de manutenção seja reduzido. Por causa da necessidade de compreender o sistema existente e analisar o impacto das mudanças no sistema, torna-se mais dispendioso acrescentar funcionalidades depois da entrega. Portanto, qualquer trabalho que favoreça na redução de custos dessa análise durante o desenvolvimento, será útil para reduzir custos de manutenção. Boas técnicas de engenharia de software, como a especificação precisa, o uso de desenvolvimento orientado a objetos e o gerenciamento de configuração contribuem para a redução dos custos de manutenção.

Os principais fatores que distinguem o desenvolvimento da manutenção e que levam a custos elevados de manutenção são:

- Estabilidade da equipe – a equipe de desenvolvimento do sistema normalmente se dispersa após a entrega do projeto, deixando a necessidade de formar uma nova equipe. Esta, por sua vez, precisa de muito esforço para compreender o sistema existente e prestar a manutenção;

- Responsabilidade contratual – o contrato para fazer a manutenção de um sistema, geralmente é separado do contrato de desenvolvimento. Contudo, existe a possibilidade da manutenção ser concedida a uma empresa diferente. Esse fator, juntamente com a instabilidade da equipe, resulta em nenhum incentivo para que a equipe de desenvolvimento escreva o software de maneira a facilitar sua modificação;
- Habilidade da equipe – a manutenção pode ser designada a um pessoal técnico mais novo, pois requer menos habilidade com o desenvolvimento. Além disso, o sistema pode ter sido escrito em uma linguagem obsoleta e o pessoal de manutenção pode não ter muita experiência com ela, havendo a necessidade de aprendê-la para fazer a manutenção. Segundo Lientz e Swason (1981) é um dos problemas da manutenção onde a demanda é competitiva para o tempo do profissional;
- Idade e estrutura do programa – à medida que os programas envelhecem, suas estruturas tornam-se mais difíceis de ser entendidas e modificadas. Além disso, estas estruturas podem ter sido otimizadas com vistas à eficiência e não à facilidade de compreensão. Ainda, suas documentações podem ter sido perdidas ou estarem inconsistentes. Pode-se ter problema para identificar a versão mais atual, que deve ser modificada, por não ter passado pelo gerenciamento de configuração.

2.1.3 Processo de Manutenção de Software

O processo a ser descrito é apresentado por Wilson de Pádua Paula Filho (2001) em uma versão simplificada e adaptada do processo recomendado na norma IEEE Std. 1219-1993 – *IEEE Standard for Software Maintenance* (IEEE, 1994).

2.1.3.1 Objetivos

Um processo de manutenção de software deve ser capaz de garantir que:

- Os pedidos de manutenção sejam documentados, de forma que possibilite identificar o que é solicitado e o que é implementado;

- Os pedidos de manutenção sejam tratados através de processo bem definido, para que seja possível identificar onde tenham ocorrido problemas;
- Seja possível identificar e manter as diversas versões de cada produto;
- Seja possível produzir um histórico das várias alterações aplicadas sobre um produto.

Wilson de Pádua (2001) aconselha o uso das práticas de Gestão de Software para atingir estes objetivos através:

- do controle dos componentes do produto como itens de Gestão de Software;
- da organização destes itens como linha de base;
- do controle e checagem de todas a alterações;
- da recuperação e auditoria de todas as modificações;
- do fácil acesso a versões oficiais, atualizadas por todos os membros da equipe.

2.1.3.2 Classificação das Solicitações de Manutenção de Software

As solicitações de manutenção de software são classificadas quanto ao tipo:

- Problema que não de manutenção – o problema relatado é erro do usuário ou defeito em parte que não seja do produto;
- Correção urgente – problema que traz perturbação significativa ao trabalho do usuário;
- Correção não-urgente – problema que pode ser contornado por um período de tempo razoável;
- Melhoria menor – aperfeiçoamento do produto, de grande utilidade que pode ser feito com custo, prazo e risco pequenos;
- Melhoria maior – aperfeiçoamento do produto, de utilidade mais distante para o usuário, que não pode ser feito com custo, prazo e risco pequenos.

2.1.3.3 Descrição do Processo de Manutenção de Software

O processo de manutenção de software é composto por cinco fases consecutivas, correspondentes à identificação, análise, desenho, implementação e testes. Descrevem-se a seguir estas fases. Ao longo destas fases aparecem inúmeros personagens distintos, que se encontram envolvidos com elas. A Figura 2.1 apresenta uma visão geral destes personagens.

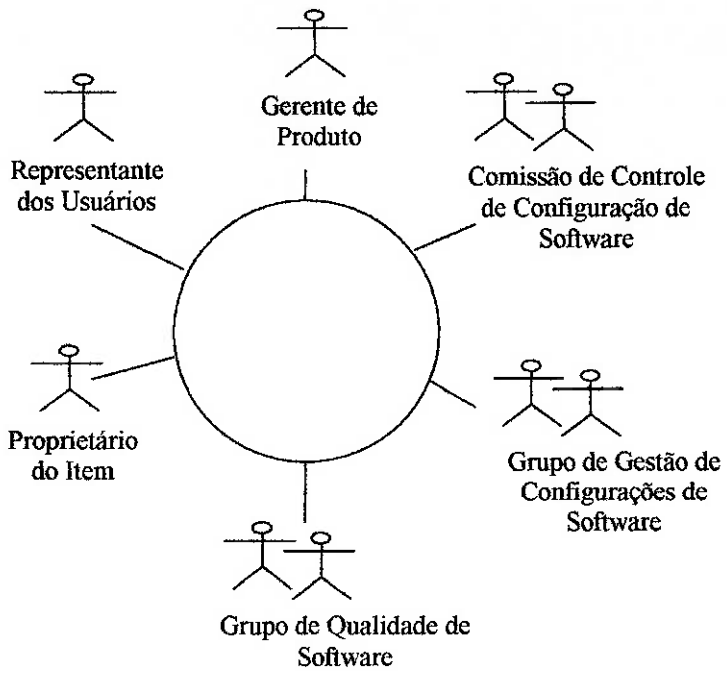

Figura 2.1 – Equipe envolvida no processo de manutenção

Os personagens que comparecem na Fig. 2.1 representam papéis bem definidos neste cenário. Estes papéis estão descritos a seguir.

- Gerente de produto – responsável pela manutenção de um produto ou grupo correlato de produtos onde toma as providências necessárias para que os procedimentos de manutenção sejam respeitados; dirige a Comissão de Controle de Configurações de Software; autoriza a liberação dos itens do respectivo produto para manutenção; acionando os proprietários destes itens; recebe os itens

modificados submetendo-os a testes, revisões e demais procedimentos cabíveis de Garantia da Qualidade; aciona as alterações das linhas de base para refletir as modificações efetuadas; providencia a reconstrução dos produtos; providencia a distribuição e a instalação dos produtos modificados no ambiente dos clientes e comunica aos clientes e usuários providências que devam tomar em conseqüência das atividades de manutenção;

- Representantes dos usuários – cada produto tem um representante dos usuários. Este, recebe as notificações de problemas enviadas pelos usuários; analisam estas notificações para verificar se requerem ações de manutenção, assistência no usuário ou outro tipo de providência e caso se trate de ações de manutenção, formalizam sua solicitação;

- Proprietários dos Itens – responsáveis por efetuar fisicamente os procedimentos de alteração dos itens que constituem o produto. Um produto pode ter um único proprietário, ou seus itens podem ficar sob a responsabilidade de profissionais especializados por item. Cada proprietário emite um parecer sobre as solicitações de manutenção dos itens sob sua alçada; analisa, desenha, implementa, testa e documenta as modificações que se façam necessárias e submetem ao gerente do produto os itens modificados;

- Comissão de controle de configurações de software – responsável por um produto ou grupo de produtos correlatos. Autorizam o estabelecimento de linhas de base e a identificação de itens de configuração dos produtos; representam os interesses de todos os que podem ser afetados por mudanças nas linhas de base; emitem parecer sobre as solicitações de manutenção dos produtos sob sua alçada; revisam e autorizam alterações nas linhas de base dos produtos em operação e autorizam a construção de produtos a partir das linhas de base;

- Grupo de gestão de configurações de software – na área de manutenção, este grupo responsabiliza-se em checar a conformidade das linhas de base dos produtos, através de Auditorias de Gestão de Configurações; administram a Biblioteca de Manutenção de Software, inclusive manutenção, análise de integridade e realização de cópias de segurança; comunicam à Gerência de Manutenção os problemas relativos à Gestão de Configurações encontrados

dentro dos produtos, para que sejam providencias suas resoluções; emitem periodicamente, relatórios sobre as atividades de Gestão de Configurações à Gestão de Configurações; verificam as providências tomadas pelos gerentes de produto para resolução dos problemas encontrados e comunicam a Gerência Executiva em casos de problemas encontrados não serem resolvidos no nível de produtos;

- Grupo de qualidade de software – é um grupo de posição independente ao projeto responsáveis pelas revisões de software com ênfase na garantia da qualidade. Estes, recebem os resultados das atividades enviadas pelo gerente do projeto e os repassam para a equipe de revisão técnica recebendo, após a revisão destes, o Relatório de Revisão Técnica. Estes relatórios são registrados pelo grupo de qualidade de software, em uma base de revisões e encaminhados para o gerente do projeto.

Fase de Identificação

Na fase de identificação, as Solicitações de Manutenção de Software são recebidas classificadas e priorizadas pelo Gerente do Produto. A solicitação pode ser aceita normalmente, aceita em regime de emergência, incorporada em um lote para processamento futuro ou rejeitada, quando se identifica que o problema não é de manutenção. Para cada tipo de classificação, o Gerente do Produto terá uma atitude correspondente.

Fase de Análise

Nessa fase, o Gerente do Produto ou alguém por ele designado, realiza uma análise de impacto da modificação solicitada, gerando um Relatório de Avaliação de Solicitação de Manutenção procurando determinar os respectivos custos e benefícios.

Baseada nesses dados, a Comissão de Controle de Configurações de Software do produto decide se a manutenção será efetuada ou não.

Se a solicitação for rejeitada, o Gerente do Produto comunica a rejeição ao Representante dos Usuários, através de Relatório de Rejeição de Alteração. Caso

contrário, se a solicitação for aprovada, o Gerente do Produto emite a Proposta de Alteração para o Grupo de Gestão de Configurações de Software, para cada Proprietário de Item afetado pela solicitação e para o Representante dos Usuários.

Fase de Desenho

Nesta fase de desenho, o Grupo de Gestão de Configurações de Software libera a linha de base do produto para os Proprietários dos Itens afetados, enviando a estes e ao Gerente do Produto o Aviso de Liberação de Produto. Estas alterações podem envolver código, desenho e testes. São identificados os itens afetados, assim como os testes de regressão que deverão se aplicados. O desenho das alterações e dos respectivos testes se refletem em novos dados para a Proposta de Alteração revisada: os itens existentes podem ser alterados, e é incluído um plano de testes específico da modificação desenhada.

Fase de Implementação

Nessa fase, os Proprietários dos Itens realizam as alterações no código, executam os testes de unidades dos módulos afetados e atualizam os modelos e documentos, quando necessário. O Gerente do Produto coordena os testes de unidade e de integração desenhados, a consolidação e revisão da linha de base provisória e a atualização dos documentos e modelos.

Fase de Testes

Na fase de testes, são realizados os testes de aceitação e regressão relativos aos itens alterados, coordenados pelo Gerente do Produto. Este, envia os relatórios das revisões e dos testes ao Grupo de Qualidade de Software, o qual realiza auditoria da qualidade da linha de base modificada provisória. O Gerente do Produto envia a linha de base modificada para o Grupo de Gestão de Configuração de Software, juntamente com um Relatório de Formalização de Alteração. O Grupo de Gestão de Configuração de Software, por sua vez, incorpora a linha de base modificada à Biblioteca de Manutenção.

Fase de Instalação

A fase de Instalação completa o ciclo de manutenção, instalando o produto alterado junto ao cliente. Nessa fase o gerente do produto entrega ao representante dos usuários as cópias dos itens que deverão ser instalados ou reinstalados junto ao cliente. O representante dos usuários providencia a instalação definitiva, com assistência do gerente do produto, conforme o necessário e o combinado, e envia ao gerente do produto um Aviso de Aceitação de Alteração.

2.1.4 Manutenção da Especificação

Após o desenvolvimento de um sistema ter sido terminado e este ter entrado em operação, quase todos os analistas e desenvolvedores são deslocados para novos projetos. Porém, o trabalho feito pelo analista de sistemas continua a ser importante pois, a especificação precisa ser mantida durante o período de vida do sistema.

De acordo com Edward Yourdon (1990), os sistemas de informações têm uma importância característica relacionada à manutenção: eles duram mais que os desenvolvedores ou os usuários que estiverem envolvidos no desenvolvimento original do sistema. Outro ponto importante, é que eles tendem a ser complexos desde o início, e a complexidade cresce durante os anos de manutenção. Por isso, é difícil alguém que não esteve envolvido no desenvolvimento do sistema original ou que não possua uma documentação dos requisitos e do projeto, entender o sistema para fornecer a manutenção.

A manutenção da documentação é a última coisa que alguém deseja fazer, e muitas vezes acaba por não ser feita, quando é dado importância apenas à correção de um problema.

2.1.4.1 Pré-Requisitos Necessários

Para se ter uma manutenção apropriada, inicialmente recomenda-se garantir que a documentação do sistema esteja completa, consistente, correta e atualizada quando este entrar em operação.

Além de se certificar que a documentação esteja correta, deve-se assegurar de que existe um mecanismo para executar modificações continuadas nesses documentos.

2.1.4.2 Como fazer

Edward Yourdon (1990) propõe que a primeira regra básica da manutenção de sistemas é examinar o impacto na especificação de requisitos do sistema de qualquer modificação proposta. Qualquer modificação deve ser ilustrada, documentada e verificada com o usuário, fazendo as modificações adequadas no modelo do sistema. Esta manutenção inicia-se pelo preenchimento de um formulário que pode ser denominado Solicitação de Alteração de Sistema.

A modificações solicitadas podem ser mínimas que só requeiram alguns minutos de trabalho ou podem exigir um tempo considerável para a realização. Independente disso, o grupo responsável pela alteração deve produzir uma Declaração de Impacto, a qual apresentará uma declaração precisa e detalhada das mudanças que terão de ser feitas na especificação do sistema para implementar a modificação proposta. Junto com este documento, deve haver uma Declaração do Impacto Econômico, o qual implicará no custo da implementação da alteração e o benefício estimado que derivará dela.

Há algumas modificações que não causam impacto na especificação do sistema tais como: a correção de um erro de programa, a modificação do código para melhorar a legibilidade ou a eficiência do sistema ou uma mudança no hardware existente ou do software de sistema – compilador, sistema operacional, sistema de gerenciamento de banco de dados. Mesmo assim deve ser emitida uma Declaração do Impacto Econômico para que o usuário e o setor de desenvolvimento de sistemas sejam informados dos custos e benefícios relacionados àquela alteração.

As modificações podem também causar atualizações em manuais, em procedimentos de operação e em vários outros componentes de sistema. Porém, o documento mais importante é a Declaração de Requisitos, o qual deve ser mantido atualizado. Sem isso, futuras modificações podem se tornar mais dispendiosas, mais consumidoras de tempo e mais penosas do que normalmente seriam.

2.2 *Extreme Programming* - XP

Segundo Kent Beck (2000), esta metodologia recebe este nome porque o termo *Extreme* enfatiza o uso extremo de práticas que se mostram funcionais, como: revisões de código, teste, simplicidade e ciclos curtos e iterativos.

Kent Beck relata que um dos principais problemas relacionados ao desenvolvimento de software é o risco. Atrasos no cronograma, projetos cancelados devido a esses atrasos, sistemas tornando-se obsoletos, alta taxa de defeitos, mudanças de requisitos e saída de importantes membros da equipe de desenvolvimento são exemplos de riscos que podem resultar no fracasso de um projeto de software.

Extreme Programming – ou XP –, é um novo modelo para o processo de desenvolvimento de software que visa alcançar duas metas almejadas pela indústria de tecnologia da informação: desenvolvimento rápido e consistente com as reais necessidades do cliente e fácil manutenibilidade, permitindo que o software seja modificado à medida que as necessidades do negócio se alteram ou ampliam.

Extreme Programming começou a ser desenvolvido em 1996 por Kent Beck no Departamento de Computação da montadora de automóveis *DaimlerChrysler*, e possui muitas diferenças em relação aos outros métodos, podendo ser aplicado a projetos com altos riscos e requisitos dinâmicos.

Comunicação, Simplicidade, *Feedback* e Coragem são os quatro lemas adotados pelos seguidores de XP, que correspondem a quatro dimensões nas quais os projetos podem ser melhorados. XP oferece condições para que os desenvolvedores possam responder de forma confiável às alterações de requisitos propostas pelos clientes, mesmo em estágios finais do ciclo de vida do processo.

Entretanto, XP não se aplica bem a todo e qualquer tipo de projeto e, como todo processo, possui algumas restrições. Para que sua aplicação seja produtiva são necessárias algumas características, entre elas:

- Grupos Pequenos – XP supõe que as equipes de desenvolvimento de um projeto possuem de 2 a 10 programadores;

- Trabalho em equipe – XP expande a equipe de desenvolvimento incluindo forte integração entre gerentes e clientes durante todo o processo de desenvolvimento;
- Testabilidade – é mister poder criar testes funcionais e unitários automatizados. Às vezes é necessário alterar o projeto do sistema para facilitar os testes;
- Produtividade – é necessária uma equipe de desenvolvimento comprometida e dinâmica para assegurar um alto grau de produtividade, que é indispensável na realização dos projetos XP.
- Agilidade na comunicação com o cliente – a metodologia de desenvolvimento enfoca a rapidez da implementação e, desta forma, a comunicação com o cliente deve ser ágil. É preciso que o cliente especialmente dedicado ao projeto possa tomar decisões rápidas para garantir o cronograma do projeto.

Desta forma, XP requer uma mudança cultural profunda, o que nem sempre é fácil alcançar. Além disso, alguns obstáculos à sua implementação podem surgir como: gerentes ou clientes que insistem em ter um conjunto completo de especificações ou um projeto detalhado antes da fase de codificação, ou ainda, sistemas com uma grande quantidade de aplicações já existentes e difíceis de serem alteradas não oferecem flexibilidade suficiente para garantir a simplicidade no código, um dos requisitos de XP.

2.2.1 Valores, Princípios e Requisitos Básicos de XP

Kent Beck (2000), apresenta um conjunto de valores, princípios e requisitos básicos que visam alcançar eficiência e efetividade no desenvolvimento de software, seguidos pelo XP. Os valores são quatro: comunicação, simplicidade, *feedback* e coragem. Nestes valores, estão fundamentados alguns princípios básicos: *feedback* rápido, simplicidade, mudanças incrementais – e apenas quando necessárias – e trabalho com qualidade. Por fim, nesses princípios se baseiam os doze requisitos básicos adotados por XP:

1) Processo de Planejamento (*Planning Process)*, também chamado de *Planning Game* – o processo de planejamento de XP permite que o "cliente XP" defina o valor

de negócio dos recursos desejados e utilize estimativas de custo fornecidas pelos programadores para decidir o que é necessário ser feito e o que pode ser adiado.

2) Pequenos lançamentos – as equipes XP colocam um sistema simples em produção com antecedência, e o atualizam freqüentemente em ciclos bastante curtos.

3) Metáforas do Sistema – as equipes XP utilizam um "sistema de nomes" e uma descrição do sistema sem a utilização de termos técnicos, para guiar o desenvolvimento e a comunicação com o cliente.

4) Projeto simples – um programa construído através do método XP deve ser o mais simples possível satisfazendo os atuais requisitos, sem a preocupação de atender outros que surgirão no futuro. O foco está em prover valor de negócio.

5) Teste – as equipes XP focalizam a validação do software durante todo o processo. Os programadores desenvolvem software escrevendo primeiro os testes, e só então o software que atenda aos requisitos desses testes. Os clientes provêm testes de aceitação para ter certeza que os recursos necessários estão sendo fornecidos.

6) Reconstrução – as equipes XP procuram aperfeiçoar o projeto do sistema durante todo o desenvolvimento, mantendo a clareza do software: sem ambigüidade, com alta comunicação, simples, porém completo.

7) Programação em dupla – os programadores XP produzem o código em duplas, ou seja, dois programadores trabalhando juntos na mesma máquina. Muitos experimentos têm mostrado que a programação em dupla produz software de melhor qualidade com um custo similar ou menor do que o produzido por programadores trabalhando individualmente.

8) Propriedade coletiva – todo o código pertence a todos os programadores. Essa característica permite que a equipe trabalhe a toda velocidade, uma vez que as alterações podem ser feitas sem atrasos, pois todos têm liberdade para fazê-las.

9) Integração contínua – as equipes XP integram e constroem o sistema de software várias vezes por dia. Isso mantém todos os programadores em sintonia e possibilita um progresso rápido.

10) 40 horas de trabalho semanal – programadores exaustos cometem mais erros. As equipes XP não trabalham por um tempo excessivo, mantendo-se, assim, mais efetivas.

11) Cliente dedicado – um projeto XP é conduzido por um indivíduo dedicado (um cliente), que determina os requisitos, atribui as prioridades, e responde as dúvidas dos programadores (relacionadas aos requisitos). Essa prática melhora a comunicação e gera menos documentos, o que, em geral, é uma das partes mais caras num projeto de software.

12) Código padrão – para que uma equipe trabalhe em dupla de forma efetiva e compartilhe a propriedade de todo o código, todos os programadores precisam escrever da mesma forma, com regras que assegurem a clareza do código.

2.2.2 O Ciclo de Vida e as Fases do Processo

A figura 2.2 representa as fases do ciclo de vida do XP.

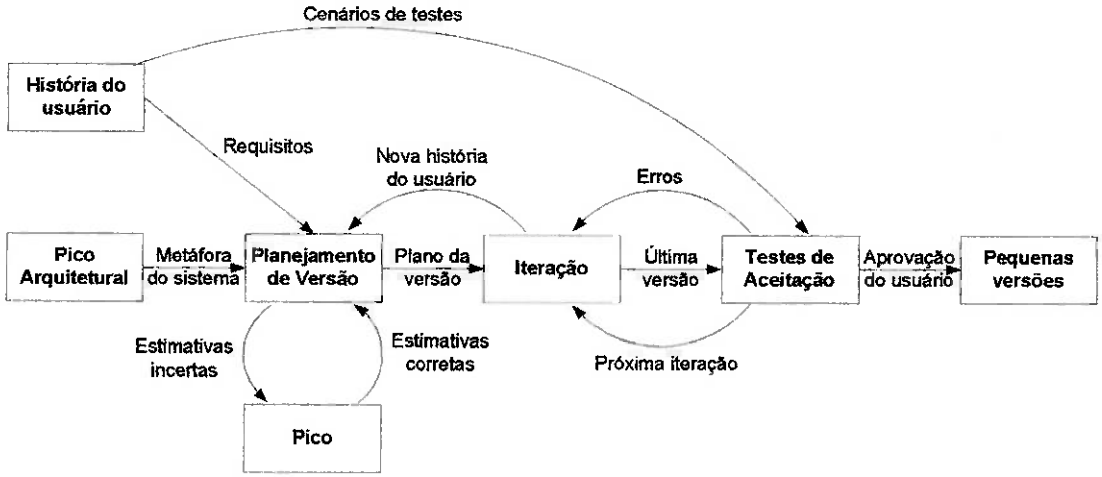

Figura 2.2 – Ciclo de vida do XP
(Figura adaptada de http://extremeprogramming.org/map/project.html)

O ciclo de vida XP é bastante curto, no entanto, esta abordagem pode fazer sentido em um ambiente onde as mudanças de requisitos do sistema são fatores dominantes.

As Histórias do Usuário - *User Stories* - são documentos escritos pelos próprios usuários para a obtenção de requisitos, para gerar as estimativas do planejamento da versão e para auxiliar na criação de cenários de testes.

Na fase Pico Arquitetural – *Architectural Spike* – são exploradas soluções para reduzir riscos de algum problema técnico, os quais são escritos na forma de metáforas, e também soluções que auxiliam as estimativas do planejamento da versão.

Em Planejamento da Versão – *Release Planning* – os requisitos do cliente são cuidadosamente coletados na medida em que são fornecidos. O planejamento consiste em estimar diversos fatores que podem afetar o desenvolvimento do software. Algumas das tarefas do planejamento incluem: decidir escopo e prioridade do projeto, estimar custos e cronogramas e criação de um plano para a entrega de uma nova versão do produto. Uma diferença entre o XP e a maioria dos modelos de processo convencionais, é que XP não define a especificação formal e completa de requisitos.

Na Iteração – *Iteration* – o desenvolvimento é efetuado de forma iterativa, ou seja, o cronograma é dividido em inúmeras iterações de uma a três semanas. Se no momento do desenvolvimento for adicionada uma nova história do usuário, a versão deve ser replanejada.

Os Testes de Aceitação – *Acceptant Tests* – são elaborados a partir das especificações do cliente e realizados a cada iteração. Somente depois que todos os erros forem corrigidos, deve-se passar para uma nova iteração. De acordo com Martin Fowler e Matthew Foemmel, um dos doze requisitos básicos de XP é o de escrever testes antes do código. Os testes em XP são divididos em duas categorias: testes unitários e testes de aceitação, também chamados de testes funcionais. Testes unitários são, em geral, escritos pelos desenvolvedores e têm a finalidade de testar uma classe individual ou um pequeno grupo de classes. Já os testes de aceitação são usualmente escritos pelos próprios clientes ou por uma equipe de testes externa, que conta com a ajuda dos desenvolvedores, e têm a finalidade de testar todo o sistema, de ponta-a-ponta. Para Ambler (2004), os testes de aceitação são considerados

artefatos de requisitos. As atividades de teste são realizadas durante todo o processo de desenvolvimento e o código é construído com o propósito de satisfazer os resultados esperados. E na medida em que um novo código é adicionado, novos testes devem ser realizados para assegurar que impactos negativos não venham a ocorrer.

As Pequenas Versões – *Small releases* – são liberadas para utilização assim que todas as iterações sejam finalizadas, ou seja, depois que todo o desenvolvimento e todos os testes sejam concluídos.

2.2.3 Algumas considerações sobre o ciclo de vida XP

Ambler (2004) agrupa as etapas de Teste, Codificação e Projeto em uma fase que chama de Iterações para Entrega, seguindo-se a ela uma fase de Produção, na qual o sistema é testado no ambiente final de produção. É nesta fase que se aplicam os testes de sistema, de carga e de instalação, podendo surgir novos defeitos, que necessitam ser consertados.

A fase de Manutenção da XP é o estado normal dos projetos, englobando todo o ciclo de vida; assim, é feita a suposição que os projetos sempre estão evoluindo.

2.2.4 Estratégias de Gerenciamento

A principal estratégia de gerenciamento em XP baseia-se na utilização de princípios básicos de negócio: entrega do produto em fases, *feedback* rápido e concreto, clareza e objetividade das necessidades do sistema e a alocação de especialistas para executar tarefas específicas.

Em muitos casos, o planejamento estratégico e a tomada de decisões são feitos de forma centralizada, isto é, são realizados por uma única pessoa responsável: o gerente. Ou então ocorre o oposto, todos tomam decisões sem adotar nenhum critério. Para balancear esses dois extremos, XP recorre novamente aos princípios a que se propõe:

- Responsabilidade aceita – o papel do gerente é salientar o que é necessário ser feito, e não distribuir tarefas;
- Trabalho com qualidade – existe uma grande diferença entre o gerente exigir que os desenvolvedores façam um bom trabalho e o gerente colaborar para que os desenvolvedores façam um trabalho ainda melhor;
- Mudanças incrementais – o gerente deve orientar a equipe durante todo o tempo, e não simplesmente apresentar um manual de regras logo no início do projeto;
- Métricas – são as principais ferramentas de gerenciamento em XP. A medida principal a ser obtida é a proporção entre o calendário atual e o tempo estimado para o desenvolvimento. A medida entre tudo que foi previsto e o que de fato ocorreu também é essencial para a melhoria no processo. No entanto, quaisquer que sejam as métricas coletadas, elas devem refletir a realidade e devem ser precisas. O gerente também deve assegurar que as métricas sejam visíveis a todos cujo trabalho está sendo medido.

2.3 *Rational Unified Proces* – RUP

O *Rational Unified Process* (RUP) é um processo de engenharia de *software* desenvolvido pela Rational Software Corporation, cujas principais características são um desenvolvimento iterativo e incremental, orientado a objetos, com foco na criação de uma arquitetura robusta, empregando análise de riscos e guiado por casos de uso durante o desenvolvimento.

O RUP foi desenvolvido para ser aplicável a uma grande classe de projetos diferentes e pode ser considerado como um *framework* genérico para processos de desenvolvimento.

Isso significa que ele deve ser configurado para ser usado eficientemente. A configuração pode ser feita de acordo com o perfil das empresas, para definir o seu processo padrão de desenvolvimento, ou mesmo, para projetos específicos – de acordo com o domínio da aplicação – e, normalmente, envolve remoção e/ou modificação de atividades do *framework*.

O RUP é composto por quatro fases – concepção, elaboração, construção e transição – cada uma com objetivos específicos. Na fase de concepção, deve-se estabelecer o escopo e a viabilidade econômica do projeto. Na elaboração, o objetivo é eliminar os principais riscos e estabelecer uma arquitetura estável, a partir da qual o sistema poderá evoluir. Na fase de construção, um produto completo é desenvolvido de maneira iterativa até que esteja pronto para ser passado aos usuários, o que ocorre na fase de transição, onde uma versão beta do sistema é disponibilizada. No final da transição pode ser iniciado um novo ciclo de desenvolvimento para a evolução do produto, o que envolveria todas as fases novamente.

Todas as fases são finalizadas com um *milestone* no qual, contratantes e contratados verificam se os objetivos da fase foram alcançados.

Cada fase pode comportar várias iterações e cada iteração, por sua vez, está organizada em um Fluxo de Trabalho - *workflows*, que descrevem o que deve ser feito em termos de atividades, responsáveis e artefatos. A figura 2.3 apresenta as fases, as iterações e o fluxo de trabalho do RUP.

O RUP fornece modelos para cada artefato e *guidelines* para a execução de suas atividades.

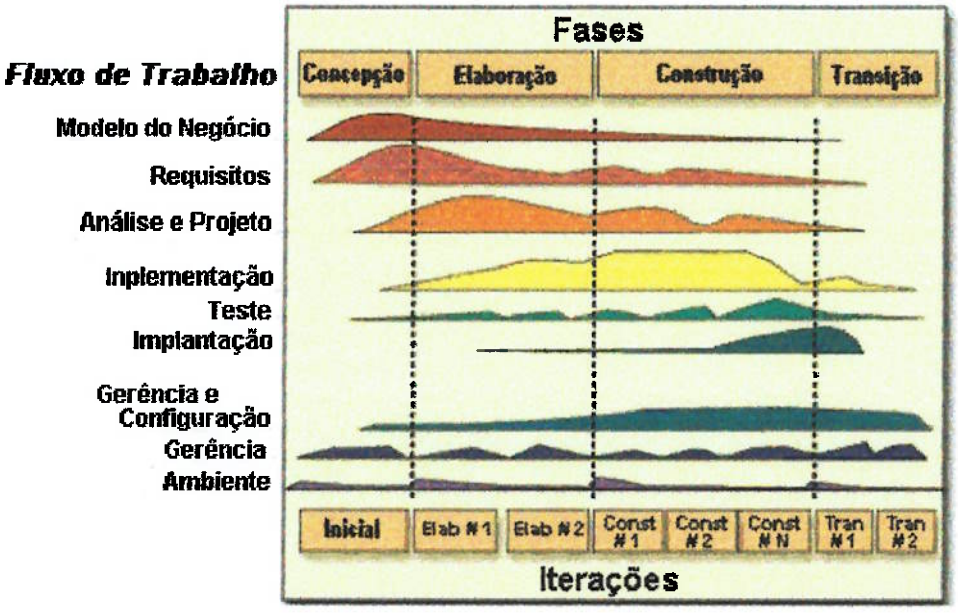

Figura 2.3 – Fases, iterações e fluxo de trabalho do RUP (figura adaptada da Rational Software Corporation – http://www.rational.com)

As iterações também são finalizadas com *milestones*, que devem controlar se foram cumpridos os objetivos específicos da iteração, como a realização de um grupo de casos de uso, por exemplo.

O Fluxo de Trabalho do RUP é composto pelo Fluxo de Trabalho de Engenharia de *Software* e pelo Fluxo de Trabalho de Suporte. O Fluxo de Trabalho de Engenharia é descrito sucintamente a seguir:

- Modelagem do Negócio – envolve o entendimento da estrutura e dinâmica da organização cliente, garantindo que clientes, usuários e desenvolvedores tenham a mesma visão da organização para a qual será feito o desenvolvimento;
- Requisitos – envolve a definição dos requisitos do sistema e de como gerenciar o escopo e mudanças de requisitos;
- Análise e Projeto – envolve a tradução dos requisitos numa especificação que descreve como implementar o sistema. A UML é utilizada para modelar o sistema;
- Implementação – envolve o desenvolvimento de código: classes, objetos etc., teste de unidades e integração de subsistemas;
- Teste – envolve a verificação do sistema como um todo, com testes de integração e conformidade com os requisitos especificados;
- Distribuição – envolve o empacotamento, distribuição, instalação e treinamento de usuários, assim como o planejamento e condução de testes beta.

O Fluxo de Trabalho de Suporte compreende as atividades necessárias para a execução do Fluxo de Trabalho de Engenharia. São elas:

- Gerência de Projeto – envolve o gerenciamento de riscos, planejamento e acompanhamento do projeto;
- Gerência de Configuração e Mudanças – envolve o gerenciamento dos artefatos gerados durante o desenvolvimento;
- Configuração do Ambiente – envolve a organização do ambiente de trabalho para a equipe do projeto e a configuração do RUP para o projeto.

2.4 Engenharia Reversa

Conforme Ian Sommerville (2003), Engenharia Reversa é o processo em que se analisa o software com o objetivo de recuperar seu projeto e sua especificação sem alterar o código-fonte.

O processo de Engenharia Reversa inicia-se com uma fase de análise, durante a qual o sistema é analisado, utilizando-se ferramentas automatizadas, a fim de descobrir sua estrutura. Engenheiros trabalham com o código-fonte do sistema e seu modelo estrutural adicionando informações obtidas mediante a compreensão do sistema. Essas informações são mantidas como um grafo direcionado, que é vinculado ao código-fonte do programa.

Os *browsers* de armazenamento de informações são utilizados para comparar a estrutura do grafo e o código e para anotar informações extras no grafo. Documentos de vários tipos, como diagramas de programa e de estrutura de dados e matrizes de rastreamento, podem ser gerados a partir do grafo direcionado. As matrizes de rastreamento mostram onde as entidades são definidas e referenciadas nos sistema. O processo de geração de documentos é iterativo, uma vez que as informações de projeto são utilizadas para melhorar ainda mais as informações contidas no repositório do sistema.

Depois que a documentação de projeto do sistema foi gerada, outras informações podem ser adicionadas ao repositório de informações para ajudar a recriar a especificação do sistema.

2.5 Considerações finais sobre os métodos

RUP e XP vêm de diferentes filosofias. RUP é um *framework* de componentes de processos, métodos e técnicas que podem ser aplicados a qualquer projeto de software específico. Devido a sua grande abrangência, é esperado que sejam feitas adaptações, seja em relação a natureza do projeto ou da organização em que está sendo utilizado, para que seja melhor aproveitado.

XP é um processo *leve*, centrado em código, para pequenos projetos cujos requisitos não possam ser bem definidos antecipadamente, ou em que se espera que hajam grandes mudanças, durante o projeto. Ele enfatiza a comunicação contínua entre usuário e membros da equipe de desenvolvimento. Como RUP, ele é baseado em iterações que incluem várias práticas como: Pequenos *Releases*, *Design* Simples, Testes e Integração Contínua.

O Processo de Manutenção estudado no item 2.1, aconselha o uso de boas técnicas de engenharia de software que contribuam para uma manutenção simplificada e de custos menores. Defende a liberação de *releases* com uma quantidade pequena de reparos, assim como uma das práticas de XP. Este processo de manutenção possui suas fases bem definidas, em contrapartida, não prevê a possibilidade de iterações em sua fase de implementação, como XP, ou em qualquer outra fase como o RUP, onde a iteração pode ocorrer uma ou mais vezes dentro de cada fase, envolvendo diversas disciplinas conforme a natureza da mesma.

A Engenharia Reversa tem com objetivo derivar o projeto ou a especificação do sistema a partir do código fonte, permitindo assim, a aplicação de uma metodologia de modo que facilite na manutenção do programa.

Abaixo citam-se algumas tarefas e produtos que são gerados nos processos apresentados, descrevendo a visão de cada processo em relação a elas:

- Identificar a Viabilidade – RUP e XP concordam que é melhor identificar o mais cedo possível se o projeto é viável, a fim de evitar o gasto de recursos valiosos em um projeto condenado. Enquanto o Processo de Manutenção faz a análise sob a solicitação;
- Plano de Aceitação do Projeto – Em um projeto XP ele aparece na forma da aceitação dos testes criados pelo usuário. Já em RUP e no Processo de Manutenção estudado, o usuário não constrói os testes, mas o critério de acesso deve ser direcionado pelo usuário, seja diretamente, ou através de outro papel, como o do Analista de Sistemas ou do Gerente do Produto;

- Modelos – Os Casos de Uso produzidos em RUP correspondem aos *User Stories* escritas pelo usuário em XP. A diferença é que o Caso de Uso é um conjunto completo de ações iniciado por um ator (alguém ou algo fora do sistema). O Caso de Uso pode conter várias *User Stories.* No Processo de Manutenção, estes modelos devem existir para orientar os desenvolvedores na manutenção e sofrer atualizações de acordo com as implementações solicitadas;
- Métricas – Há três aspectos para serem medidos no projeto de software: tempo, tamanho e defeitos. RUP provê diretrizes sobre o que pode ser medido e como medir além de oferecer exemplos de métricas. Já XP provê métricas simples usadas para determinar o progresso e estimar a conclusão de tarefas, fazendo um forte uso de uma pequena quantidade de métricas. O Processo de Manutenção, também possui métricas que podem ser úteis para avaliar a facilidade da manutenção.

Capítulo 3 – Investigação do Processo Vigente na Empresa

Neste capítulo estuda-se o processo de desenvolvimento de software utilizado na empresa. O objeto principal deste processo é um software de Call Center, destinado a apoiar as atividades de suporte a clientes de uma empresa financeira.

Utilizando-se a estrutura do RUP como base, serão investigados neste capítulo: o processo, a arquitetura do sistema de Call Center, a formação da equipe de trabalho, os documentos envolvidos, o fluxo de documentos e de informações e a ferramenta de apoio.

3.1 Visão geral do processo de manutenção na empresa

O desenvolvimento de software se dá através de um processo continuado. Estando já o software em operação, pode-se caracterizar tal processo como sendo de manutenção.

O processo de manutenção é semelhante ao processo de Ian Sommerville (2003), iniciado por um conjunto de pedidos de alterações por parte da Equipe de Processos do cliente. Um novo *release* do sistema é planejado. Durante esse planejamento, todas as mudanças propostas – reparo de defeitos, adaptação de plataforma e novas funcionalidades – são consideradas e, então, é tomada uma decisão sobre quais mudanças devem ser implementadas na próxima versão do sistema. As mudanças são implementadas e validadas e uma nova versão do sistema é liberada. Depois disso, o processo se repete com um novo conjunto de mudanças propostas para a nova versão. A Figura 3.1 mostra uma visão geral desse processo.

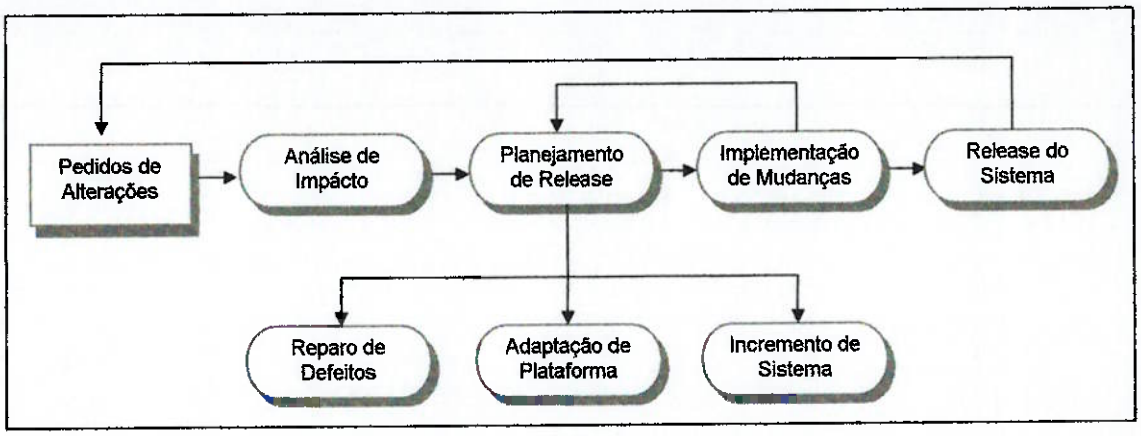

Figura 3.1 – Visão Geral do Processo de Manutenção, segundo Ian Sommerville (2003)

3.2 As pessoas envolvidas – a equipe de projeto e manutenção

Neste processo, várias pessoas diferentes se encontram envolvidas. Em princípio, as pessoas e equipes são:

- Equipe de Processos – composta por pessoas do cliente com conhecimento do negócio da empresa. Esta equipe é que solicita as mudanças a serem feitas no sistema.
- Gerente de Manutenção de Software – responsável pelas seguintes atividades:
  - tomada das providências necessárias para que sejam respeitados os procedimentos de manutenção;
  - contato com o cliente para definir as solicitações prioritárias e a data da nova versão;
  - manutenção da Gerência Executiva, com conhecimentos sobre a situação da equipe de desenvolvimento e do cliente.
- Coordenadores – responsáveis pelas seguintes atividades:
  - contato com o cliente para definir as solicitações prioritárias e a data da nova versão;
  - providências quanto à distribuição das tarefas aos analistas e desenvolvedores;

  - garantia de liberação das solicitações no prazo estimado – ou negociacão, com o cliente, de um novo prazo, caso ocorra a liberação não proceda.

- Analistas – responsáveis por:
  - analisar a especificação do cliente;
  - desenvolver a maquete;
  - estimar prazos;
  - testar a implementação.

- Desenvolvedores – recebem as solicitações dos coordenadores ou dos analistas; analisam, implementam, testam, implantam as solicitações recebidas e submetem informações relativas a um posicionamento sobre a manutenção, aos respectivos coordenadores ou analistas.

- Equipe de Banco de Dados – responsável pela administração do Banco de Dados. Analisa qualquer implementação necessária no Banco de Dados e, na real necessidade, fazem a devida manutenção.

A Equipe de desenvolvimento é constituída por:

- Um Gerente de Manutenção de Software;
- Dois Coordenadores;
- Quatro Analistas;
- Dezenove Desenvolvedores.

3.3 Documentos envolvidos

O processo de manutenção descrito na Fig. 3.1 envolve a circulação de um conjunto de documentos, que estão descritos a seguir.

- *Documento de solicitação:* documento preenchido pelo cliente em um sistema de apoio às manutenções, como requisito para a manutenção evolutiva, a qual tem como objetivo aperfeiçoar o software, implementando novos requisitos, ou manter a funcionalidade do sistema, melhorando sua estrutura e seu desempenho.

Neste documento, devem ser preenchidos: o nome do solicitante, o projeto a sofrer alteração, a data de abertura e a descrição da implementação;

- *Documento de Ocorrência:* é um documento em que se descreve a necessidade de uma manutenção corretiva, ou seja, para qualquer defeito notificado pelo cliente, ou até mesmo pela equipe de desenvolvimento, é aberta uma demanda para a correção deste. No documento de ocorrência devem ser preenchidos: o nome do solicitante, o projeto a sofrer correção, a data de abertura e a descrição do defeito;
- *Especificação do cliente:* contém a necessidade do cliente que o sistema deverá contemplar. Este documento é gerado pela Equipe de Processos, quando uma solicitação possui uma complexidade maior;
- *Maquete:* Documentação desenvolvida com base na Especificação do cliente, contendo o cronograma de desenvolvimento e implantação e as regras de negócio, no formato de um português estruturado;
- *Controle de Lógicas por Swap:* Documentação preenchida pelo desenvolvedor para cada *swap* a ser realizado (semelhante a um *Check List*). Contém o número e a descrição da demanda resolvida, o projeto em questão, a lógica e as telas envolvidas na implementação, nomes de arquivos e as datas de disponibilização da alteração em cada ambiente;
- *Controle de Demandas por Swap:* É de responsabilidade do coordenador. Contém a lista das demandas a serem disponibilizadas em cada *swap*. Discrimina o número, o tipo e a descrição da demanda, o desenvolvedor responsável, o solicitante e a situação em que a demanda se encontra (Parada / Em desenvolvimento / Em testes de desenvolvimento / Em testes em ambiente de Pré-Produção);
- *Manual do Usuário:* Desenvolvido pela Equipe de Processos.

3.4 Fluxo de Documentos e fluxo de informações

Documentos e informações necessitam fluir através da equipe para que as solicitações possam ser devidamente documentadas e agilizadas. Documentos e informações possuem seu próprio fluxo.

3.4.1 Fluxo de Documentos

A Figura 3.2 demonstra o fluxo dos documentos entre as pessoas da equipe de desenvolvimento e o cliente, representado pela Equipe de Processos.

As solicitações demandadas pela Equipe de Processos, são encaminhadas ao coordenador de uma equipe de desenvolvimento (a) o qual as prioriza juntamente com a Equipe de Processos, montando o documento de *Controle de Demandas por Swap*. O coordenador encaminha as solicitações (b) para um analista ou desenvolvedor, dependendo da complexidade da solicitação. Caso seja uma solicitação acompanhada de um documento de *Especificação do Cliente*, o analista retornará uma *Maquete* (c) para aprovação da Equipe de Processos. Se esta *Maquete* for aprovada (d), retorna para o analista que por sua vez a encaminha ao desenvolvedor (e) o qual, após a manutenção, preencherá o *Controle de Lógicas por Swap* (f) que deverá ser entregue ao coordenador após a realização do Swap.

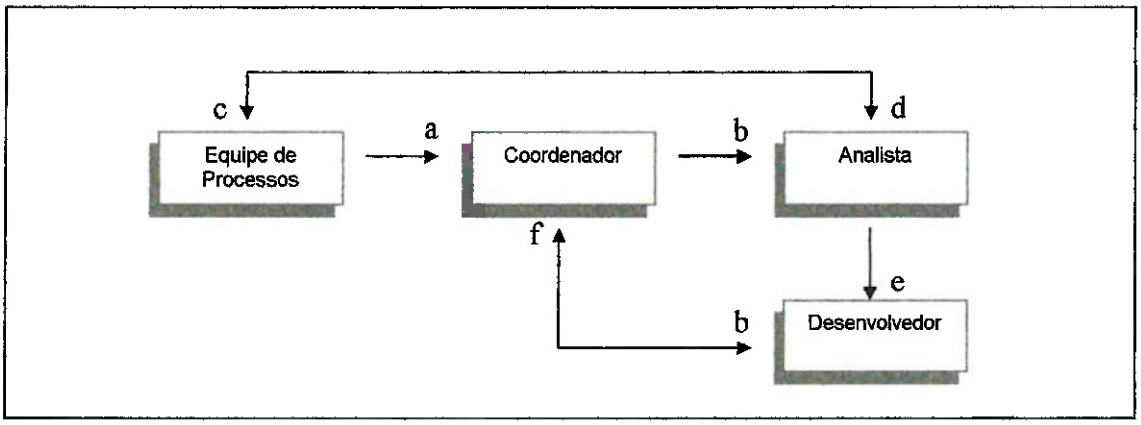

Figura 3.2 – Representação do Fluxo de Documentos

3.4.2 Fluxo de Informações

A Figura 3.3 demonstra o fluxo de informação que ocorre entre as pessoas da equipe de desenvolvimento e o cliente, representado pela Equipe de Processos.

O responsável pela solicitação da Equipe de Processos especifica sua necessidade (a) para o Analista ou para o Desenvolvedor encarregado pela demanda em questão. Este processo se origina com a abertura de uma solicitação pela Equipe de Processos via sistema. Se a manutenção requerida for complexa, a Equipe de Processos irá gerar um documento de especificação o qual será enviado por e-mail. Caso a complexidade tenha requerido o Analista, este fará a análise da manutenção e aprovando-a, encaminha a solicitação e a especificação, se houver, ao Desenvolvedor (b). Este, por sua vez, comunica ao Analista a transição de etapas da solicitação, tais como, em desenvolvimento ou testes em desenvolvimento ou testes em ambiente de homologação(c). Caso contrário, o próprio Desenvolvedor estuda a solicitação e entra em contato diretamente com a Equipe de Processos, na existência de dúvidas (d). Na etapa de testes, o Desenvolvedor comunica à Equipe de Processos a liberação da demanda (e). A Equipe de Processos testa a implementação e em caso de erro (f), a Equipe de Processos contacta diretamente o Desenvolvedor. Para a atualização em ambiente de Produção (g), o Analista ou o Desenvolvedor comunica ao Coordenador o sucesso da manutenção em ambiente de desenvolvimento, confirmando a liberação para a nova versão.

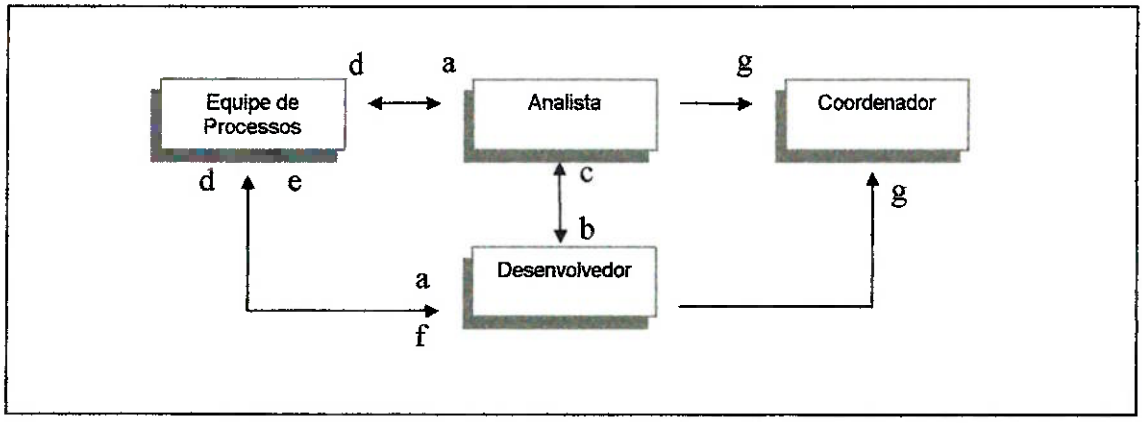

Figura 3.3 – Representação do Fluxo de Informações

3.5 Arquitetura do sistema e sua evolução

A arquitetura de um sistema baseado em bancos de dados é fortemente influenciada pelo sistema básico computacional sobre o qual o banco de dados é executado. O sistema utilizado na empresa da autora tem uma arquitetura baseada em Banco de Dados Centralizados, que é executado sobre um único sistema computacional (sistema centralizado), e que não interage com outros sistemas.

3.5.1 Gestão De Dados

O acesso a dados, se dá através de um Gerenciador de Banco de Dados Oracle e de um Banco de Dados em *mainframe*. Os dados localizados no *mainframe* são acessados via transação.

Uma transação de *input* é enviada com dados obrigatórios a um CICS – sistema de software que pode manipular e armazenar as entradas de muitas fontes diferentes – e este, apresenta a transação ao banco de dados no *mainframe* e recebe como resposta uma transação de *output*.

O dados do Banco de Dados Oracle são acessados diretamente com a ferramenta utilizada para desenvolver o sistema (Edge).

O gerenciamento do Banco de Dados Oracle está sob responsabilidade da Equipe de Banco de Dados. A manutenção no sistema que exija uma modificação no Banco de Dados – desde alteração de arquitetura do Banco de Dados, até alteração de um simples dado – é repassada à Equipe de Banco de Dados a qual analisa essa alteração e questiona a sua necessidade. O acesso aos desenvolvedores se restringe apenas à consulta dos dados, com o uso de *login* e senha.

Os testes das manutenções solicitadas, são realizados em dois ambientes:

- Um ambiente de desenvolvimento, onde há um sistema que aponta para um Banco de Dados Oracle de desenvolvimento, contendo dados de testes;
- Um ambiente de Homologação, onde existe um sistema espelho do sistema de Produção, mas com as alterações a serem implantadas, o qual aponta para um Banco de Dados Oracle de Produção, contendo dados reais.

Após a validação dos testes pela Equipe de Processos no ambiente de Homologação com dados reais, as implementações são repassadas ao ambiente final, o de Produção, o qual também aponta para o Banco de Dados de Produção.

3.5.2 Interface

A interface apresentada ao usuário é do tipo gráfico, desenvolvida com o auxílio da ferramenta utilizada, Edge, com a qual se implementa um sistema de gerenciamento de janelas e que, também permite a comunicação com o servidor. É uma interface integrada no sistema usado.

Os *layouts* das interfaces são definidos pela Equipe de Processos. Estes vêm especificados, quando existentes, no documento de Especificação do Cliente, relativo a uma manutenção solicitada.

3.5.3 Prototipação

O processo de prototipação é utilizado quando se deseja detalhar os requisitos de software definidos pelo cliente. Em outros casos, pode ser usado para avaliação de uma tecnologia a ser utilizada ou para certificar-se da forma que a interação homem-máquina pode assumir.

O protótipo tem como objetivo fornecer uma visão prévia, ao cliente, da implementação solicitada, podendo detectar falhas antes de sua codificação.

A utilização do protótipo na empresa da autora, não é freqüente. Este só é requerido no desenvolvimento de uma atividade de maior amplitude, a qual consiste em uma manutenção evolutiva e exige prazos maiores para seu desenvolvimento. O protótipo apresentado é reutilizado a partir de sua aprovação.

3.5.4 Controle de Versões

O controle de versão é realizado apenas com a determinação de datas, quando serão liberadas as implementações testadas e aprovadas pela Equipe de Processos. A identificação da versão é visualizada no sistema através de uma data especificada na tela inicial do sistema. Fica armazenada somente, a versão que se tornou a penúltima.

3.6 Ferramenta Utilizada

A ferramenta de trabalho utilizada é uma ferramenta específica para Call Center. Esta é nomeada como ferramenta Edge e foi toda desenvolvida na linguagem C e, gera um código interno com extensão .obj. É uma ferramenta orientada a eventos.

3.7 Avaliação sobre o processo atual de manutenção

O processo atual de manutenção deixa a desejar, exigindo que as fases fossem melhor definidas, como:

- Análise – é feita superficialmente, não possuindo uma firme progressão desde os aspectos de modelagem de alto nível – como diagramas –, aos aspectos de modelagem de níveis mais baixos – como o desenvolvimento de especificações de processos e de dicionário de dados;
- Desenvolvimento – não apresenta o uso de técnicas de programação;
- Testes – não há o uso de estratégias de testes para validar todos os processos que envolvem a manutenção implementada;
- Implantação – não há um controle de versão documentado.

Neste processo de manutenção, há uma grande rotatividade de elementos da equipe, tais como admissão e demissão de Gerentes de Projeto, Coordenadores, Analistas e Desenvolvedores. Por isso, nota-se a necessidade de documentações como especificação de requisitos do sistema, especificação de processos e de diagramas de Entidades e Relacionamentos no momento de atender às solicitações, podendo a sua inexistência chegar a comprometer os prazos estimados para implantação.

Capítulo 4 – Proposta para Melhoria do Processo

Nos capítulos anteriores deste trabalho apresentou-se: uma visão geral do processo de manutenção, algumas metodologias, a importância da qualidade de processo e o estudo da situação atual da empresa da autora. Neste capítulo, será apresentada uma proposta para melhor se trabalhar no projeto continuado deste sistema.

A proposta é baseada no conteúdo exposto nos capítulos anteriores e constitui o núcleo da contribuição desta pesquisa.

Conforme apresentado no capítulo anterior, a documentação existente é elaborada apenas para relatar as manutenções efetuadas e a dificuldade encontrada é a falta de documentação adequada que facilite o entendimento geral do sistema para realizar de forma mais fácil tais manutenções, bem como permitir a aplicação de uma metodologia que viabilize o processo de manutenção.

Levando em consideração essa dificuldade, é inicialmente proposto um pequeno Processo para a Engenharia Reversa, cuja contribuição é a elaboração de uma documentação atualizada do sistema existente tornando-o mais expressivo e de fácil entendimento, utilizando alguns modelos da Análise Essencial (GANE;SARSON, 1982; YOURDON, 1990).

A escolha pela utilização da Análise Essencial se dá com base nas características da ferramenta usada na empresa da autora, a qual é orientada a eventos, e à natureza do sistema sendo considerado, centrado em dados.

4.1 Processo para a Engenharia Reversa

O objetivo do processo de Engenharia Reversa é identificar no sistema os elementos que compõem os modelos da Análise Essencial. Essas informações podem ser obtidas no código fonte, única informação disponível sobre o programa.

Considerando-se os modelos da Análise Essencial, devem ser extraídos da estrutura do código elementos tais como o Diagrama de Contexto do sistema, supostos eventos, supostas entidades externas, entidades de dados e seus atributos e

relacionamentos. Os documentos referentes aos modelos serão então criados, auxiliando no processo de manutenção de futuras alterações solicitadas.

Sugere-se que a geração do Modelo Essencial seja feita em paralelo ao processo de manutenção por uma dupla formada por um Analista e um Desenvolvedor, exclusivamente destacados para esse trabalho. Estes se responsabilizariam pela geração do Modelo Essencial da parte do sistema já operante. A manutenção posterior de algum documento já criado ou a geração de novas documentações procedentes de alguma solicitação do processo de manutenção, ficaria sob responsabilidade dos Analistas responsáveis pela solicitação, os quais devem manter estreita comunicação com a dupla mencionada, para não gerar divergências na interação do Modelo Essencial.

Os documentos propostos para a geração são:

- Lista de Eventos;
- Diagrama de Contexto;
- Diagrama de Estruturas de Dados – DED e
- Diagrama de Fluxo de Dados - DFD

Estes quatro passos são discutidos a seguir.

4.1.1 Identificar os Eventos

Levantar eventos é, basicamente, identificar fatos que ocorrem no meio ambiente que interage com o sistema e que exigem uma resposta do mesmo. Esta resposta pode ser o armazenamento de uma informação ou a produção de um resultado destinado ao ambiente. Deve-se analisar o ambiente do sistema e registrar todo fato que, a princípio, pareça determinar uma ação do sistema. Cada evento deve ser descrito por uma oração que expresse uma idéia completa. A oração é uma construção gramatical que deve possuir um sujeito, um verbo e um objeto. Exemplo de eventos do sistema citado no capítulo anterior são:

1. Cliente efetua pagamento de conta de consumo.

2. Cliente efetua DOC.

Após a identificação dos eventos, a Lista de Eventos poderá ser construída. Para cada evento identificado deverá constar nessa lista: o fluxo de estímulo – entrada – e sua origem, a resposta – ação – que o sistema deve dar ao evento e o fluxo de saída produzido pelo sistema com seu respectivo destino - saída. A tabela 4.1 representa uma Lista de Eventos, conforme esta proposta.

Tabela 4.1 – Lista de Eventos

Nº.	Evento	Descrição	Estímulo	Resposta	Saída
1	Cliente efetua pagamento de conta de consumo	O cliente pode realizar o pagamento de qualquer conta de consumo.	Cliente informa o número do código de barras da conta de consumo	Conta de consumo paga	Dados de pagamento registrados no banco de dados. Emissão do comprovante de pagamento
2	Cliente efetua DOC	Quando o cliente transfere um valor para uma conta cujo banco (e o titular da conta destino) são diferentes da conta de origem, realiza-se um DOC.	Cliente informa os números das contas de origem e destino, e o valor a ser transferido	DOC efetuado.	Valor debitado da conta origem e creditado na conta destino. Emissão do comprovante de transferência

4.1.2 Determinar o Diagrama de Contexto do Sistema

O Diagrama de Contexto é a representação gráfica do Ambiente do Sistema. Nele, o sistema como um todo é representado por um círculo (que representa um processo), com o seu nome. Os usuários, fornecedores e/ou receptores de informações, são representados por retângulos (entidades externas), rotulados pelo nome do agente externo e as informações trocadas são as setas identificadas pelo nome do pacote de informações que flui entre o sistema e seus usuários. Contudo, deve-se identificar os usuários que interagem com o sistema, os dados que ele informa ao sistema e os que ele obtém como retorno, para gerar o Diagrama de Contexto. A Figura 4.1 representa um trecho de um Diagrama de Contexto, no caso, o correspondente ao evento 1 da Tabela 4.1.

Nem todo evento é sinalizado por um fluxo de dados. Os eventos temporais, por exemplo, não possuem um fluxo de dados de entrada como estímulo. Nem todo evento possui um fluxo de saída correspondente. Um evento pode ocasionar apenas o armazenamento de um dado. Eventos diferentes podem estar associados a um mesmo fluxo de dados.

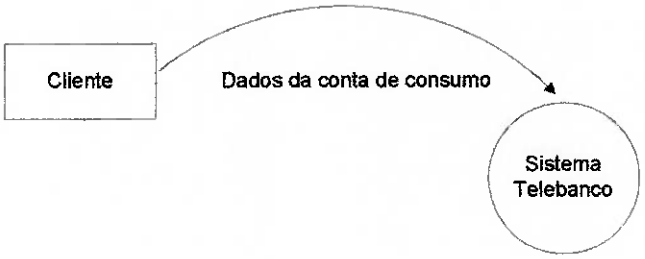

Figura 4.1 – Diagrama de Contexto

Os fluxos de dados podem ser documentados em um documento complementar denominado Dicionário de Dados, onde cada fluxo seria decomposto em seus componentes atômicos, cujos atributos seriam registrados neste mesmo dicionário.

4.1.3 Identificar as Entidades, seus Atributos e Relacionamentos

As entidades, atributos e relacionamentos serão identificados através dos comandos de abertura e acesso ao banco de dados. Esta identificação resultará na geração do Diagrama de Estrutura de Dados - DED.

O trabalho de geração do DED será benéfico também para uma triagem do que realmente está sendo usado no banco de dados. Será possível a identificação de tabelas que não são usadas e assim, eliminá-las. A detecção de atributos ou entidades duplicados poderá implicar na unificação destes. Contudo, o DED será gerado sem redundância e com uma estrutura real.

Deve ser levado em consideração que o trabalho de reestruturação do banco de dados, influenciará nas alterações de lógicas, procedimentos e funções do sistema. Entretanto, o Banco de Dados é responsabilidade da Equipe de Banco de Dados e

qualquer alteração proposta está fora de controle deste trabalho, cabendo à equipe de desenvolvimento apenas fazer sugestões sobre a reestruturação.

Considerando-se a arquitetura do sistema centrada em dados, deve-se considerar que a reestruturação dos dados pode implicar em necessidade de reestruturação mais ampla, exigindo não só a alteração dos dados como também alterações de código fonte, uma vez que os algoritmos estão intimamente ligados a seus dados.

4.1.4 Diagrama de Fluxo de Dados – DFD

De acordo com Yourdon (1990) o Diagrama de Fluxo de Dados é um modelo que nos permite imaginar um sistema como uma rede de processos funcionais, interligados por "dutos" e "tanques de armazenamento" de dados.

Para a montagem do DFD é necessário identificar no sistema:

- Processos: procedimentos em que uma ou mais entradas são convertidas em saídas. Geralmente o processo é descrito com uma única palavra ou sentença simples e representado por um círculo;
- Fluxos: representam o movimento de fragmentos ou de pacotes de informações de um ponto para outro do sistema (os "dutos"). O fluxo é representado por uma seta;
- Depósitos: modelam uma coleção de dados em repouso (os "tanques"). Para o sistema em estudo, representam os arquivos do banco de dados já identificados na construção do DED. A representação de um depósito na modelagem essencial são duas linhas paralelas;
- Terminadores: são entidades externas com as quais o sistema se comunica. Tipicamente, são pessoas ou um grupo de pessoas. São, graficamente, representados por retângulos.

A Figura 4.2 representa um DFD, o que expressa o tratamento do evento 1 da tabela 4.1.

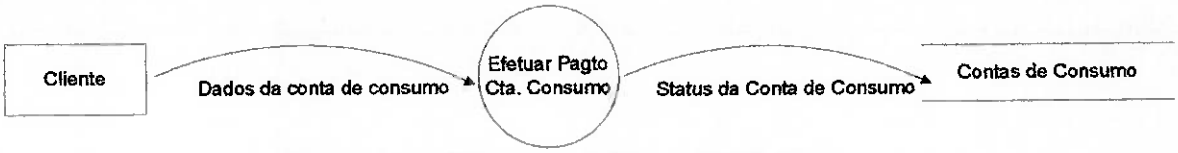

Figura 4.2 – Diagrama de Fluxo de Dados

4.2 Programa de Garantia de Qualidade de Software

A proposta para a garantia da qualidade do software consiste em procedimentos, técnicas e ferramentas para garantir que um produto atenda a padrões, que devem ser definidos pela empresa, durante o ciclo de manutenção do software. A qualidade será obtida durante o processo de manutenção com o objetivo da geração de resultados esperados pelo cliente.

O programa de Garantia de Qualidade baseia-se nas atividades de revisão e inspeção as quais auxiliarão, respectivamente, no entendimento do projeto e na garantia da qualidade do produto, de acordo com os padrões da organização.

4.2.1 Criação de um Grupo de Qualidade de Software

Primeiramente deve-se buscar apoio da gerência para a implantação do processo de melhoria de software.

Sugere-se compor o Grupo de Qualidade de Software com analistas da equipe de manutenção e da equipe de Processos do Cliente que possuam, respectivamente, alta capacidade técnica e do negócio.

O Grupo de Qualidade de Software terá por responsabilidade criar e gerenciar vários processos de melhoria das atividades em curso, fazer as revisões e inspeções da documentação gerada e realizar a auditoria da qualidade da linha de base modificada.

O Grupo de Qualidade de Software deve começar o processo de melhoria pela divulgação de sua existência, definição de seu escopo de atuação e seus objetivos.

Isto servirá para obter a cooperação dos demais elementos da equipe de desenvolvimento para as atividades que se seguirão em busca de melhorias.

4.2.2 Qualidade na Modelagem e nos Resultados dos Produtos

Como mencionado no início deste capítulo, há a necessidade de modelagem dos processos atuais para facilitar o entendimento do sistema, beneficiando-se a manutenção. Para conferir maior qualidade a este processo de manutenção, definir-se-ão atividades, métodos, práticas e transformações que se utilizam para desenvolver e manter um software e os produtos a ele associados como, por exemplo, planos do projeto, documentos de projeto, código, casos de teste e manuais de usuário.

Tendo como objetivo tornar o processo de manutenção melhor definido e possibilitando ao software ser implementado de forma mais consistente, permite-se que a organização se torne mais madura, obtendo pontos valiosos em áreas chaves dos estágios do Modelo CMM.

4.3 Processo de Manutenção

Nos itens seguintes descrevem-se as atividades que deverão ser realizadas no processo proposto, juntamente com os documentos manipulados em cada fase. Note-se que nesta proposta mantém-se o ciclo de vida original da empresa, entretanto, incluem-se novos documentos no fluxo existente, como, por exemplo, os referentes ao Modelo Essencial e o Documento de Planejamento de Testes.

4.3.1 Fase de Identificação

Na fase de identificação, as solicitações de manutenção de software são recebidas pelos Coordenadores, através do Documento de solicitação que pode vir acompanhado por uma Especificação do cliente, ou do Documento de ocorrência, abertos pela Equipe de Processos ou até mesmo por integrantes da equipe de desenvolvimento. Estas demandas são classificadas e priorizadas pelo Gerente de Manutenção de Software juntamente com os Coordenadores e a Equipe de Processos

quando também, definem uma data para a próxima versão. Assim como XP trabalha com pequenas *releases,* este processo de manutenção libera novas versões com periodicidade de 15 dias.

Após a priorização das demandas, os Coordenadores montam o documento Controle de Demandas por *Swap,* o qual é um artefato que atende a área chave Planejamento do Projeto de Software, no Nível 2 do CMM; sua finalidade é planejar e documentar as atividades e os compromissos do projeto de software.

Entrada(s):

- Documento de solicitação / Documento de ocorrência – Aberto pela Equipe de Processos. O *status* deste documento é "pendente".
- Especificação do cliente – Quando for uma manutenção evolutiva, a Equipe de Processos encaminhará a especificação referente ao Documento de solicitação aberto.

Saída(s):

- Controle de Demandas por *Swap* – Os Coordenadores relatarão as solicitações priorizadas.

4.3.2 Fase de Análise e Projeto

Nessa fase, o Coordenador encaminha o Documento de ocorrência ou o Documento de solicitação e a Especificação do cliente, se houver, a um Analista. Este, coletará os requisitos que forem necessários para completar a especificação do cliente, realizará uma análise de impacto da modificação solicitada, irá gerar um Cronograma para a liberação da solicitação - que poderá coincidir ou não com a data pré-definida para a nova versão. Também, solicitará o documento de Planejamento de Testes à Equipe de Processos, pois, assim como a metodologia XP recomenda, um bom procedimento preparatório para a atividade de testes é escrever o que deve ser testado, e atualizará a documentação do sistema ou gerará novos documentos que compõem o Modelo Essencial, caso exista necessidade, notificando o Grupo de Qualidade de Software para realizar a devida inspeção.

Para o Desenvolvedor, o Analista especificará como a manutenção deverá ser implementada no sistema utilizando a Especificação do cliente, quando houver, mais sua análise, gerando o documento Maquete.

Os documentos Especificação do cliente, Maquete e o Modelo Essencial são artefatos que contribuem com a área chave Gestão de Requisitos, do Nível 2 do CMM, a qual especifica que os requisitos devem ser documentados e revisados. O documento Cronograma atende a área chave Planejamento de Projeto de Software, também do Nível 2 do CMM, onde consta um planejamento para a realização do trabalho.

Entrada(s):

- Documento de solicitação – Aberto pela Equipe de Processos. O *status* deste documento ainda é "pendente".
- Especificação do cliente – Quando for uma manutenção evolutiva, a Equipe de Processos encaminhará a especificação referente ao Documento de solicitação aberto.

Saída(s):

- Cronograma – Consta o período de desenvolvimento, de testes e de implantação, o qual será encaminhado para o cliente;
- Maquete – Este documento é elaborado pelo Analista e encaminhado ao Desenvolvedor para efetuar a manutenção;
- Documento de solicitação – O *status* deste documento é alterado para "em Desenvolvimento";
- Planejamento de Testes;
- Modelo Essencial (atualizado).

4.3.3 Fase de Implementação

Nessa fase, os Desenvolvedores, já com as devidas especificações em mãos, realizam as alterações no código e executam os testes unitários dos módulos afetados, seguindo o especificado no documento Planejamento de Testes. Em paralelo, os Desenvolvedores montam o documento Controle de Lógicas por *Swap*, onde constam os nomes das lógicas, telas, arquivos e depósitos de dados que sofreram alterações. O processo de implementação da manutenção é acompanhado pelos Coordenadores com o objetivo de prover visibilidade adequada do progresso real, permitindo que sejam executadas ações efetivas quando o desempenho da manutenção desvia significativamente dos planos de software.

Entrada(s):

- Documento de solicitação – o *status* do sistema é "em Desenvolvimento";
- Maquete;
- Cronograma;
- Planejamento de Testes – Roteiro de testes a ser seguido após terminar a produção de um artefato, para verificar se este foi corretamente produzido.

Saída(s):

- Documento de solicitação – o *status* deste sistema é alterado para "em teste";
- Controle de Lógicas por *Swap*.

4.3.4 Fase de Testes

Na fase de testes, são realizados os testes de aceitação relativos aos itens alterados, coordenados pela Equipe de Processos do cliente. Estes, seguem também seu Planejamento de Testes validando a manutenção solicitada.

O Planejamento de Testes depois de efetuados os testes, deve ser encaminhado ao Grupo de Qualidade de Software para o processo de revisão, no qual os resultados obtidos serão comparados com os previstos na Especificação do cliente. Esta

validação é uma das atividades da área chave Garantia da Qualidade, do Nível 2 do CMM que revisa a disponibilização dos resultados ao cliente.

Entrada(s):

- Documento de solicitação – o *status* deste sistema é "em Teste";
- Planejamento de Testes.

Saída(s):

- Documento de solicitação – o *status* deste sistema é alterado para "Testes OK".

4.3.5 Fase de Implantação

Nessa fase completa-se o processo de manutenção. Os Desenvolvedores seguem o que foi especificado no documento de Controle de Lógicas por *Swap* para finalizar a manutenção atualizando o ambiente de produção. Este documento é entregue para o Coordenador da equipe. O Coordenador compara o documento Controle de demandas por *Swap,* onde constam as solicitações e ocorrências que serão liberadas, com o documento Controle de Lógicas por *Swap,* o qual possui também as solicitações e ocorrências com seus respectivos nomes de lógicas, nomes de telas, arquivos e depósitos de dados alterados.

Entrada(s):

- Documento de solicitação – o *status* deste sistema é "Testes OK";
- Controle de demandas por *Swap*;
- Controle de Lógicas por *Swap.*

Saída(s):

- Documento de solicitação – o *status* deste sistema é alterado para "Finalizado".

Todo o processo em que uma manutenção é submetida, está sendo acompanhada através da evolução do status dos documentos contudo, o Documento de solicitação e

o Documento de ocorrência contribuem com a área Acompanhamento e Supervisão do Projeto de Software, do Nível 2 do CMM.

As revisões e auditorias realizadas pelos Coordenadores e pela Equipe de Garantia da Qualidade são atividades previstas na Garantia da Qualidade de Software com o objetivo de subsidiar a manutenção do sistema e os Desenvolvedores envolvidos garantindo a satisfação do cliente.

Para facilitar o trabalho entre os Desenvolvedores compartilhando a propriedade do todo o código, sugere-se que a Equipe de Qualidade de Software gere um documento padronizado para codificação, com regras que assegurem a clareza do código, para que os programas sejam escritos da mesma forma e compreensíveis a todos. O Código padrão e a Propriedade coletiva são requisitos do XP.

4.4. Conclusões

Neste capítulo procurou-se estabelecer de forma rigorosa a definição do processo de manutenção na empresa, visando atingir áreas chave do modelo CMM. Entretanto, é necessário considerar que este modelo envolve atividades de natureza gerencial e que somente a melhoria dos processos técnicos de desenvolvimento e manutenção não seria suficiente para propiciar a elevação de nível da empresa. Esforços deveriam ser realizados no sentido de permitir o planejamento e gerenciamento dos projetos com base na experiência adquirida em projetos anteriores. Isto equivale a dizer que a empresa deve, antes de tudo, preparar-se para organizar uma base de dados de históricos de projetos, fundamental ao planejamento correto de cada projeto que se inicia. Outros fatores, como a análise de riscos, controle de versões e implantação de métricas também são importantes e deveriam ser considerados, caso o objetivo primeiro deste trabalho tivesse sido a elevação de maturidade.

O objetivo principal, entretanto, foi realizar o planejamento das atividades visando a obtenção de documentação do sistema, cuja falta torna-se cada vez mais crucial ao processo de manutenção. Os modelos sugeridos – Análise Essencial – tiveram a escolha fortemente influenciada pela ferramenta utilizada no desenvolvimento, que não pode ser desprezada em detrimento de outras metodologias. Por outro lado, a

adoção desta metodologia pode ser apoiada por inúmeras ferramentas CASE disponibilizadas gratuitamente na Internet, cuja aquisição não traria nenhum ônus adicional à empresa.

Capítulo 5 – CONCLUSÕES

5.1 Considerações Finais

Embora sabendo que não existam soluções fáceis e instantâneas para os problemas de manutenção de software, deve-se considerar que:

- Para aplicar uma metodologia em um sistema já implantado, deve-se levar em conta as ferramentas utilizadas para se aplicar uma metodologia compatível;
- Antes de começar a melhorar o processo de manutenção de software, deve-se definir o processo - gerenciamento de processos -, para que se possam identificar as causas básicas da baixa qualidade de versões e projetos de software e identificar a porção do software que causa mais desperdício, retrabalho e, portanto, aumento de custos à medida que o software evolui;
- A aplicação de todos os métodos, ferramentas ou procedimentos ligados à área de manutenção de software podem se tornar efetivos, se aplicados a um processo bem controlado, disciplinado e "limpo";
- Os programas de produtividade e qualidade de software devem estar voltados às questões técnicas do ciclo de desenvolvimento e manutenção dos sistemas, e principalmente, aos aspectos administrativos e humanos do trabalho;
- A maioria dos problemas associados à manutenção de software pode estar ligada a deficiências na fase de planejamento e desenvolvimento do software. A falta de controle e disciplina, e a despreocupação com a manutenibilidade futura dos sistemas nas atividades de desenvolvimento quase sempre acarretam problemas durante a manutenção do software.

5.2 Conclusão Final

Foi proposta nesta monografia uma abordagem para a melhoria de um processo de manutenção, com o uso de engenharia reversa, para a criação de Modelagem Essencial do sistema já implantado e que não possui especificação, do uso de alguns

requisitos da disciplina *Extremme Programming* e de referências aos artefatos exigidos pelas áreas chaves do Nível 2 do Modelo CMM.

A contribuição deste trabalho se dá pela aplicação de uma metodologia que define um processo de manutenção organizado por fases à empresa da autora, o qual auxiliará no procedimento adequado para o atendimento de uma solicitação de manutenção do cliente. A metodologia também beneficia nos padrões da documentação pré-definida que deve ser gerada ou atualizada, facilitando na compreensão do sistema em futuras análises. As técnicas de engenharia reversa têm como objetivo melhorar a confiabilidade e manutenibilidade do sistema.

As referências feitas ao modelo CMM são um incentivo para a adoção de um modelo de qualidade que provê o gerenciamento - com a adequada visibilidade - do processo, que está sendo utilizado pelo projeto continuado, e dos produtos, que estão sendo construídos.

A revisão e a auditoria de produtos de software e atividades para verificar se os mesmos estão cumprindo os procedimentos e padrões adotados na empresa contribuem com o subsídio da manutenção do sistema para alcançar os resultados esperados com maior qualidade do produto e, conseqüentemente, contribuindo para a satisfação do cliente.

REFERÊNCIAS

IEEE Standards Board. **Glossary of software engineering terminology** . New York: IEEE, 1983. (ANSI/IEEE Std.).

IEEE Standards Board. **Standard for software user documentation**. New York: IEEE,1987. (IEEE Std. 1063-1987)

BECK, K. **Extreme programming explained:** embrace change. Reading: Addison Wesley, 2000.

CORREIA, M. G. **Gestão de processo de desenvolvimento de projetos de software em um ciclo de vida acelerado.** 2003. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo. São Paulo, 2003.

DONOVAN, W. **Extreme programming:** a gentle introduction. Disponível em: <http://www.extremeprogramming.org >. Acesso em: 12 dez. 2003.

FOUNER, R. **Guia prático para desenvolvimento e manutenção de sistemas estruturados.** 2.ed. Trad. De Flávio Deny Steffen. São Paulo: Makron Books, 1994.

FOWLER, M.; MATTHEW, F. **Continuous integration**. Disponível em: <http://www.martinfowler.com/articles/continuousIntegration.html#N222>. Acesso em: 12 dez. 2003.

GANE, T.; SARSON, C. **Análise estruturada de sistemas**. Rio de Janeiro: LTC, 1983.

JEFFRIES, R. **Xprogramming.com**: an extreme programming resource. Disponível em: <http://www.xprogramming.com/ >. Acesso em: 12 dez. 2003.

LIENTZ, B. P. ; SWASON, E.B. Problems in application software maintenance.

Communications of the ACM, v.24, n. 11, p. 763-769. Nov. 1981.

MCMENAMIN, S. M.; PALMER, J. F. **Análise essencial de sistemas**. São Paulo: Makron Books, 1984.

SOMMERVILLE, I. **Engenharia de software**. 6.ed. São Paulo: Addison Wesley, 2003.

PAULA FILHO, W. DE P. **Engenharia de software**. Rio de Janeiro: LTC, 2001.

PRESSMAN, R. S. **Software engineering:** a practitioner's approach. 5.ed. Rio de Janeiro: McGraw-Hill, 2003.

RATIONAL SOFTWARE CORPORATION. **Rational Unified Process** – best practices for software development teams. Disponível em: < http://www.rational.com >. Acesso em: 12 dez. 2003.

RATIONAL SOFTWARE CORPORATION. **Reaching levels CMM levels 2 and 3 with the Rational Unified Process**.(White Paper). 2000. Disponível em: <http://www.rational.com> Acesso em: 16 Jan. 2003.

YOURDON E. **Análise estruturada moderna**. Rio de Janeiro: Campus, 1990.