ALISON CLARK ALVES

Aplicação de um Processo de Manutenção de Software para pequenos Projetos

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do Curso de MBA em Engenharia de Software

Área de concentração:
Engenharia de Software.

Orientadora:
Profª. : Jussara Pimenta Matos, MSc.

São Paulo
2004

FICHA CATALOGRÁFICA

Alves, Alison Clark.
Aplicação de um Processo de Manutenção de Software para Pequenos Projetos. São Paulo 2004.
61 p.

MBA (Monografia) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais. Curso de engenharia de software.

1. Engenharia de software. 2. Processo de software. 3. Manutenção de software.

I.Universidade de São Paulo. Escola Politécnica. Departamento de engenharia de computação e Sistemas digitais II.t

À minha esposa que apoiou com muito carinho e paciência em todos os momentos deste trabalho.

AGRADECIMENTOS

À professora Jussara Pimenta Matos pela orientação, dedicação e incentivo no desenvolvimento dessa monografia.

À minha esposa Alessandra pelo apoio, incentivo e companheirismo.

Aos meus pais por estarem sempre me apoiando.

À amiga Luzia pelo incentivo.

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

RESUMO

Este trabalho apresenta um estudo de processo de manutenção de software visando à qualidade e o gerenciamento de pequenos projetos. A proposta envolve o uso de engenharia reversa para a geração dos artefatos necessários para um sistema já implantado, de acordo com a manutenção.

ABSTRACT

This work presents a study about maintenance software process look at quality and management of small projects. The proposal embraces the use of reverse engineering to generate essential artifacts of a production system according with maintenance.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ALI	– Arquivo lógico interno.
AIE	– Arquivo e interface externa.
CE	– Consulta externa.
CASE	– Computer-Aided Software Engineering.
EE	– Entrada externa.
IEEE	– Institute of Electrical and Electronics Engineers.
IEC	– International Electrotechnical Commission
IFPUG	– International Function Point Group
ISO	– International Organization for Standardization.
FIREMAN	– Software de gerenciamento de chamados.
LOC	– Lines of Code.
PCU	– Pontos por caso de uso.
SE	– Saída externa.
UML	– Unified Modeling Language.

1. INTRODUÇÃO

1.1 Objetivo

O objetivo deste trabalho é fornecer um processo de manutenção de software adequado a pequenos projetos, prevendo os benefícios advindos de se possuir uma organização de atividades, facilitando a compreensão da equipe das metas a serem alcançadas, incluindo os artefatos gerados.

1.2 Motivação

Manutenção de software é sempre um desafio. No desenvolvimento de um software, em muitos casos, é difícil para as empresas adotarem um processo bem definido. Conseqüentemente, existe uma alta demanda de manutenção, falta de qualidade do produto de software gerado, incluindo também aspectos de gerenciamento de projeto.

A principal motivação para a elaboração desta monografia é ter a oportunidade de implantar a proposta apresentada neste trabalho, na empresa em que tem a necessidade de um processo de manutenção.

1.3 Estrutura do trabalho

O primeiro capítulo apresenta uma breve introdução, contemplando o objetivo e a motivação para o desenvolvimento do trabalho.

No segundo capítulo, são descritos o modelo em cascata e a engenharia reversa. O modelo em cascata é utilizado para identificação das fases, isto é, engenharia de

sistemas, análise, projeto, implementação e teste no processo de manutenção de software para pequenos projetos. A engenharia reversa é utilizada na proposta do processo para gerar os artefatos necessários, com o objetivo de atender as melhorias do processo.

No terceiro capítulo são descritas as métricas de estimativas de tamanho de software, sendo consideradas a análise por ponto de função e ponto por caso de uso. A métrica de software é de fundamental importância para a melhoria da qualidade de software.

No quarto capítulo é apresentado o processo de manutenção de software atual dentro da empresa escolhida como modelo. Nesse capítulo são apresentados os fluxos de trabalho, os artefatos utilizados, os papéis de cada profissional, as ferramentas utilizadas e uma avaliação do processo de manutenção.

No capítulo 5, com base nos modelos pesquisados, nas métricas de software estudadas e na análise do processo atual, é apresentada uma proposta contemplando, um processo de manutenção de software com melhorias voltadas à qualidade e gerenciamento de projetos, onde são indicados: a utilização de engenharia reversa do módulo a ser desenvolvido, a descrição dos casos de uso envolvidos, a elaboração dos diagramas em notação UML relacionados nos casos de uso descritos e a aplicação de métrica de software. Finalmente, no capítulo 6 é descrita a conclusão do trabalho.

2. MODELOS PESQUISADOS

2.1 Modelo em Cascata

O modelo em cascata, algumas vezes chamado de ciclo de vida clássico ou modelo seqüencial linear, sugere uma abordagem sistemática e seqüencial para o desenvolvimento de software, que começa no nível de sistema e progride através da análise, do projeto, da codificação, do teste e da manutenção.

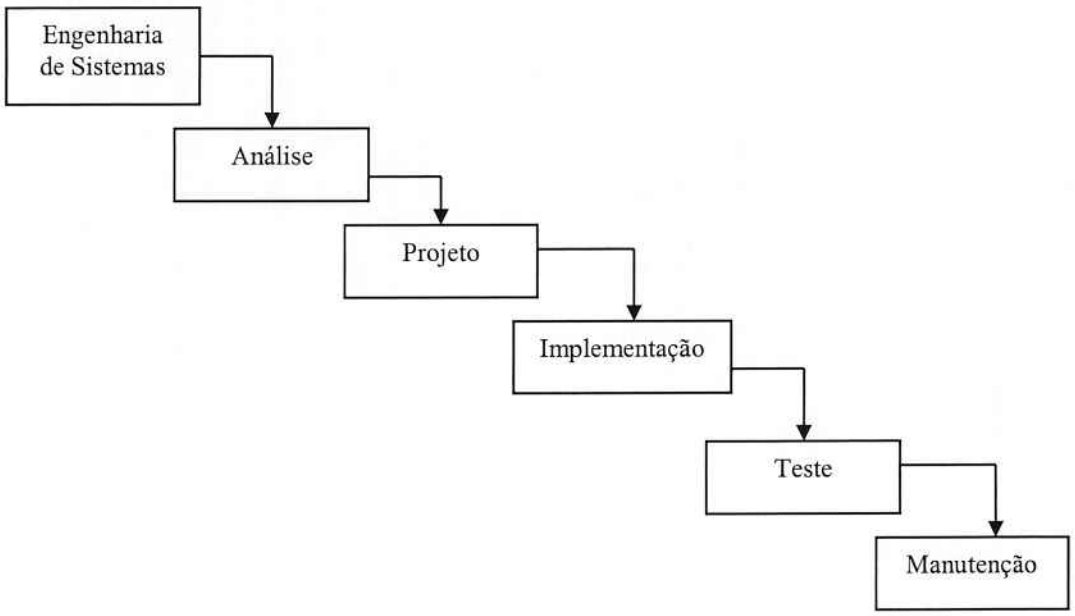

Figura 2.1 – O Modelo em Cascata

(PRESSMAN, 2001)

A fig. 2.1 ilustra o modelo em cascata aplicado na engenharia de software, abrangendo as seguintes atividades:

- **Engenharia de Sistemas**: o trabalho começa pelo estabelecimento de requisitos para todos os elementos do sistema, com base nas regras do negócio, e depois pela alocação de algum subconjunto desses requisitos para o software. Essa visão de sistema é essencial quando o software precisa interagir com outros elementos tais como, hardware, pessoas e base de dados. A engenharia e a análise de sistemas incluem um conjunto de necessidades no nível de sistema com um pouco de projeto e análise de alto nível. A engenharia da informação inclui um conjunto de necessidades a nível estratégico e no nível da área de negócios.
- **Análise**: o processo de definição de requisitos é intensificado e focalizado especialmente no software. Para entender a natureza do programa a ser construído, o engenheiro de software tem de conhecer tanto do domínio da informação do software quanto das funções necessárias, do comportamento, e das interfaces, principalmente. Os requisitos do software são documentados e revistos com o cliente.
- **Projeto**: o projeto de software é, na verdade, um processo de múltiplos passos que enfoca quatro atributos distintos do programa: a estrutura de dados, a arquitetura do software, as representações das interfaces e os detalhes procedimentais (algorítmicos). O processo de projeto traduz os requisitos para uma representação do software, que pode ser avaliada quanto à qualidade, antes que a codificação tenha início. À semelhança dos requisitos, o projeto é documentado e torna-se parte da configuração do software.
- **Implementação**: o projeto deve ser codificado, utilizando-se ferramentas de apoio para desenvolvimento do projeto.
- **Teste**: uma vez codificado, o teste do programa tem início. O processo teste focaliza os aspectos lógicos internos do software, garantindo que todos os comandos sejam testados, e os aspectos externos funcionais isto é, conduz

testes para descobrir erros e garantir que entradas definidas produzirão resultados reais, que concordam com os resultados exigidos.

- **Manutenção**: uma modificação acontece quando erros são encontrados, quando o software precisa ser adaptado para acomodar mudanças no seu ambiente externo (p. ex., uma modificação necessária por causa de um novo sistema operacional), ou quando o cliente deseja melhorias funcionais ou de desempenho. A manutenção do software reaplica cada uma das fases precedentes a um programa existente.

O modelo em cascata é o mais antigo e é o paradigma para a engenharia de software amplamente usado. Todavia, a crítica do paradigma levou até mesmo seus adeptos a questionar sua eficácia.

Entre os problemas que são algumas vezes encontrados quando o modelo em cascata é aplicado estão:

- Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Apesar de o modelo em cascata acomodar interação, o faz indiretamente. Como resultado, modificações podem causar confusão à medida que a equipe de projeto prossegue.
- Em geral, é difícil para o cliente estabelecer todos os requisitos explicitamente. O modelo em cascata exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de vários projetos.
- Não é possível obter uma versão executável do programa até o projeto terminar, fazendo com que o cliente aguarde um tempo maior para utilizar o sistema. Um erro grosseiro pode ser desastroso, se não for detectado até que o programa executável seja revisto.

Numa análise de projetos reais, foi descoberto que a natureza linear do ciclo de vida em cascata leva o projeto a um estado de bloqueio, nos quais alguns membros da equipe de projeto precisam esperar que outros membros completem as tarefas

pendentes. Na realidade, o tempo gasto em espera pode exceder o tempo gasto no trabalho produtivo. O estado de bloqueio de um projeto tende a ocorrer mais no início e no fim de um processo seqüencial linear.

Cada um desses problemas é real. Todavia, o paradigma do ciclo de vida em cascata tem um lugar importante e bem definido no trabalho de engenharia de software. Ele fornece um gabarito no qual os métodos de análise, projeto, codificação, teste e manutenção podem ser situados. Ele continua sendo um modelo amplamente usado para a engenharia de software. Apesar de ter pontos fracos, é significativamente melhor que uma abordagem aleatória para o desenvolvimento de software (PRESSMAN, 2001).

2.2 Engenharia Reversa

Engenharia reversa de software é o processo de análise de um programa, num esforço para representá-lo em uma abstração mais alta do que o código-fonte. A engenharia reversa é um processo de recuperação de projeto. As ferramentas de engenharia reversa extraem informação do projeto de dados, arquitetural e procedimental, para um programa existente.

O nível de abstração de um processo de engenharia reversa e as ferramentas usadas para executá-lo referem-se à sofisticação da informação de projeto, que pode ser extraída de código-fonte. Idealmente, o nível de abstração deveria ser tão alto quanto possível, isto é, o processo de engenharia reversa deveria ser capaz de originar representações de projeto procedimental (uma abstração de baixo nível), informação de programa e estrutura de dados e de controle (num nível de abstração relativamente alto) e modelos entidade-relacionamento (num alto nível de abstração). À medida que o nível de abstração aumenta, é fornecida informação ao engenheiro de software que permite um entendimento mais fácil do programa.

A complementação de um processo de engenharia reversa refere-se ao nível de detalhe que é fornecido num nível de abstração. Na maioria dos casos, a complementação diminui à medida que o nível de abstração aumenta. Por exemplo, dada uma listagem de código-fonte, é relativamente fácil desenvolver uma representação completa do projeto procedimental. Representações simples do fluxo de dados também podem ser originadas, mas é muito mais fácil desenvolver um conjunto completo de diagramas de fluxo de dados ou de modelos entidade relacionamento. O processo de engenharia reversa é apresentado na fig. 2.2.

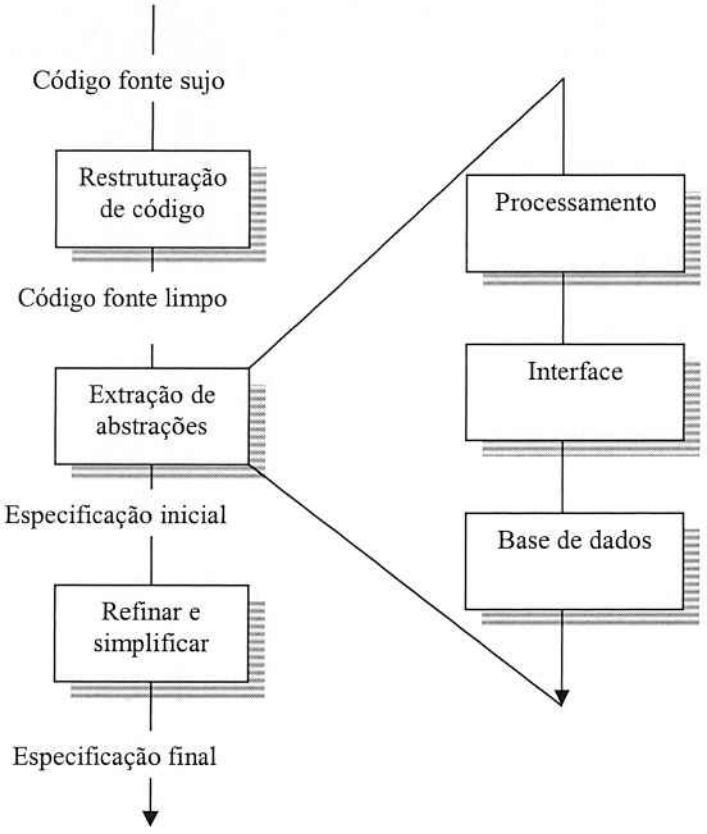

Figura 2.2 – O Processo de Engenharia Reversa
(PRESSMAN, 2001)

A primeira atividade da engenharia reversa começa com uma tentativa de entender, e depois extrair, abstrações procedimentais representadas pelo código-fonte. Para entender as abstrações procedimentais, o código é analisado em diferentes níveis de abstração: sistema, programa, componente, padrão e declaração.

A funcionalidade global de todo o sistema de aplicação deve ser entendida antes que ocorra um trabalho mais detalhado de engenharia reversa, estabelecendo um contexto para posterior análise e possibilita o entendimento de aspectos de interoperabilidade entre as aplicações. Cada um dos programas que constitui o sistema de aplicação representa uma abstração funcional num alto nível de detalhes. É criado um diagrama de blocos representando a interação entre essas abstrações funcionais. Cada componente realiza alguma subfunção e representa uma abstração procedimental definida. É criada uma narrativa de processamento para cada componente.

A complexidade se torna maior quando o código de um componente é considerado. O engenheiro procura seções do código que representem padrões procedimentais genéricos. Em quase todo o módulo, uma seção do código prepara os dados para processamento, uma seção do código diferente faz o processamento e outra seção prepara os resultados do processamento para mandar para fora do módulo. Dentro de cada uma dessas seções, podemos encontrar pequenos padrões, por exemplo, validação de dados e verificação de limites, que freqüentemente ocorrem dentro da seção de código que prepara dados para processamento.

Para grandes sistemas, a engenharia reversa é geralmente conseguida usando uma abordagem semi-automática. Ferramentas CASE são usadas para analisar a semântica do código existente. A saída desse processo é então passada para ferramentas de reestruturação e para engenharia avante, para completar o processo de reengenharia.

A engenharia reversa de dados ocorre em diferentes níveis de abstração. No nível de programa, estruturas de dados internas do programa devem ser freqüentemente submetidas à engenharia reversa como parte de um esforço de reengenharia global. No nível de sistema, estruturas de dados globais freqüentemente são submetidos a reengenharia para acomodar novos paradigmas de gestão de base de dados. A engenharia reversa das estruturas de dados globais atuais fornece o arcabouço para a introdução de uma nova base de dados para todo o sistema.

Técnicas de engenharia reversa para os dados internos de um programa focalizam uma definição de classes de objetos. Isso é conseguido pelo exame do código do programa com o objetivo de agruparem variáveis do programa relacionadas. Em muitos casos a organização dos dados no código identifica tipos abstratos de dados. Por exemplo, estruturas de registro, de arquivos, de listas e outras estruturas de dados freqüentemente fornecem um indicador inicial de classes.

Sugere-se a seguinte abordagem para a engenharia reversa de classes:

- Identifique sinais e estruturas de dados locais dentro do programa, que registram informação importante a respeito das estruturas de dados globais (p. ex., arquivo ou base de dados).
- Defina as relações entre sinais e estruturas de dados locais e as estruturas de dados globais. Por exemplo, um sinal pode ser ativado quando um arquivo está vazio; uma estrutura de dados local pode servir como depósito que contém os últimos 100 registros obtidos de uma base de dados central.
- Para toda variável (no programa) que represente uma tabela ou arquivo, liste todas as outras variáveis que têm conexão lógica com ela.
- Esses passos permitem ao engenheiro de software identificar classes no programa, que interagem com as estruturas de dados globais.

Independentemente de sua organização lógica e estrutura física, uma base de dados permite a definição de objetos de dados e apóia alguns métodos para estabelecer relações entre objetos. Assim sendo, passar de um esquema de base de dados para outro, através de reengenharia, exige um entendimento dos objetos e seus relacionamentos existentes.

Os seguintes passos podem ser usados para definir o modelo de dados existente como precursor de um novo modelo de base de dados através de reengenharia:

- Construir um modelo de dados inicial. As tabelas definidas como parte do modelo podem ser obtidas revendo registros. Os itens contidos nos registros tornam-se atributos de uma tabela.
- Determinar os candidatos importantes. Os atributos são examinados para determinar se são usados para apontar para um outro registro ou tabela. Os que servem como ponteiros tornam-se candidatos importantes.

Uma vez conhecida a informação definida nos passos precedentes, uma série de transformações pode ser aplicada para mapear a antiga estrutura da base de dados numa nova estrutura.

Interfaces com o usuário, sofisticadas, tornaram-se uma exigência de produtos e de sistemas de todos os tipos. Assim sendo, o desenvolvimento das interfaces com os usuários, tornou-se um dos tipos mais comuns de atividade de reengenharia. Mas antes que uma interface com o usuário possa ser reconstruída, deve ocorrer a reengenharia reversa.

Para entender totalmente uma interface com o usuário existente, a estrutura e o comportamento da interface devem ser especificados.

São citados dois pontos tão logo a engenharia reversa da interface do usuário se inicie:

- Definir quais são as ações básicas que a interface deve processar (p. ex., pressões de teclas e cliques de mouse);
- Descrever de maneira compacta a resposta comportamental do sistema e essas ações.

A maior parte da informação necessária para criar um modelo comportamental pode ser obtida pela observação da manifestação externa da interface existente. Mas a informação adicional necessária, para criar o modelo comportamental precisa ser extraída do código.

É importante notar que uma interface substituta não pode espelhar exatamente a interface antiga (de fato, pode ser radicalmente diferente). Freqüentemente, vale a pena desenvolver novas metáforas de interação. Por exemplo, uma interface antiga exige que um usuário forneça um fator de escala (que varia de 1 a 10) para reduzir ou ampliar uma imagem gráfica. Uma interface submetida a reengenharia poderia usar uma barra de rolagem e um mouse para conseguir a mesma função (PRESSMAN, 2001).

2.3 Considerações Finais

Conforme apresentado, no modelo em cascata a manutenção do software reaplica cada uma das fases precedentes a um programa existente e continua sendo um modelo amplamente usado para a engenharia de software (PRESSMAN, 2001). Portanto a proposta é utilizar o modelo cascata e suas fases, isto é, engenharia de sistemas, análise, projeto, implementação e teste, no processo de manutenção de software para pequenos projetos.

Conforme apresentado, engenharia reversa de software é o processo de análise de um programa, num esforço para representá-lo em uma abstração mais alta do que o código-fonte. A engenharia reversa é um processo de recuperação de projeto (PRESSMAN, 2001). Portanto, a proposta contempla a utilização de engenharia reversa, para gerar os artefatos necessários, com o objetivo de atender as melhorias no processo de manutenção para pequenos projetos.

3. MÉTRICAS DE ESTIMATIVAS DE TAMANHO DE SOFTWARE

3.1 Introdução

O objetivo desse capítulo é apresentar alguns dos conceitos relacionados à medição e às métricas de estimativa de tamanho de software, considerando a análise por ponto de função e ponto por caso de uso.

Eficiência, entrega do produto no prazo, dentro do orçamento e com um nível de qualidade desejado pelo cliente são características que influenciam no sucesso de organizações de desenvolvimento de software no mercado atual, globalizado e competitivo de tecnologia da informação. Neste sentido, ressaltam-se a importância de mecanismos de acompanhamento e de avaliação do progresso do processo, do projeto e do produto.

Estes mecanismos, normalmente baseados em informações qualitativas e quantitativas coletadas através de medições realizadas durante o projeto, são recursos essenciais na gestão de desenvolvimento de software. Para o gerente do projeto, essas medidas, também conhecidas como métricas, permitem melhor entendimento do processo de produção e do controle do projeto de software. Desta maneira, as medições fornecem informações que permitem a determinação de pontos fortes e fracos dos processos e do produto, indicam ações corretivas e propiciam avaliações de impactos de tais ações. Enfim, garantem a qualidade do processo e dos produtos obtidos (BASILI; CALDIERA; RUMBAUCH, 1994) (FENTON, 2000) (PFLEEGER, 1997).

A necessidade de medidas que informem a eficiência do desenvolvimento e minimizem os fracassos de projetos, principalmente em relação às falhas no cronograma e nas estimativas realizadas, demandaram a realização de estudos na área

de estimativa de tamanho de software, que resultaram na proposição de diversas métricas ou métodos de medição de processo, do projeto e de produtos de software.

3.2 Medição de Software

Estimativa de tamanho de software é um processo pelo qual uma pessoa ou um grupo de pessoas estima o tamanho de um produto de software (MCPHEE, 1999). Geralmente, o tamanho tem impacto na solução técnica e na gestão do projeto já que, estimativas imprecisas podem levar ao fracasso do projeto (ROSS, 1999).

As primeiras métricas de estimativa de tamanho de software surgiram em 1950/1960 e se basearam no tamanho físico de linhas de código (LOC-"Lines of Code") (FENTON, 2000). Esta métrica considera o software sob a perspectiva da estrutura interna e é aplicada nas fases finais do projeto.

Duas vantagens são destacadas no uso de LOC:

- A possibilidade de fazer a estimativa automaticamente.
- A facilidade de uso de dados históricos, pois, grande parte dos dados de estimativas existentes, foram medidos através de LOC.

Várias desvantagens são destacadas no uso de LOC:

- Ambigüidade, pois a métrica trata de abstrações não textuais;
- A falta de significado da medida para o usuário final;
- Depende do grau de reuso;
- Depende da linguagem de programação;
- Pode ser cinco vezes superior a uma outra estimativa, devido às diferenças das técnicas de medição de linhas em branco, linhas de comentário, declaração de dados e linhas de instruções;

- Penaliza programas pequenos e bem projetados, que não se adaptam às linguagens não procedimentais;
- É de difícil obtenção na fase inicial de planejamento (FENTON, 2000) (PFLEEGER, 1997).

LOC foi uma métrica bastante utilizada até meados de 1970. A partir daí surgiram diversas linguagens de programação e conseqüentemente a necessidade de outras formas de estimar o tamanho de software.

Atualmente, são várias as métricas de estimativa de tamanho existentes e grandes as dificuldades para selecionar a mais apropriada para o tamanho de um projeto de software de uma organização. As principais métricas foram desenvolvidas com base nas funções de software tais como: análise de pontos de função e pontos de caso de uso (GARMUS; HERRON, 2000) (MCPHEE, 1999).

A análise de pontos de função é uma das primeiras métricas a medir o tamanho do software com alguma precisão, é a mais utilizada no mercado, tornando-se padrão internacional em 2002 através da norma ISO/IEC 20.926 (AGUIAR, 2003) (DEKKERS, 2003). Atualmente, o mapeamento da análise por ponto de função para estimativa de projetos de software orientados a objetos tem sido largamente discutido na literatura.

O ponto de casos de uso foi definido para estimar projetos orientados a objetos, com base na mesma filosofia da análise por ponto de função e no processo Objectory (KARNER, 1993), onde foi desenvolvida a técnica de diagramação para o conceito de casos de uso. Posteriormente, Ivar Jacobson (1996) desenvolveu um método orientado a objeto, centrado em casos de uso, técnica largamente utilizada na indústria para descrever e capturar requisitos funcionais de software (RIBU, 2001). Considerando-se que o modelo de casos de uso foi desenvolvido para capturar os requisitos baseados no uso e na visão dos usuários, faz sentido basear a estimativa de

tamanho e recursos de projetos de software nos casos de usos (DAMODARAN, 2004).

Como a análise por ponto de função é a métrica mais usada no mercado e provê estimativas mais precisas à medida que se obtém mais informações do projeto e o ponto de casos de uso foi criado para estimar projetos orientado a objeto com base no modelo de casos de uso, essas métricas foram selecionadas como objeto de estudo deste trabalho. Dessa forma, as características relevantes e o processo de contagem são apresentados a seguir.

3.3 Análise por Ponto de Função

Métricas de software orientadas a função usam uma medida da funcionalidade entregue pela aplicação como valor da normalização. Como funcionalidade não pode ser medida diretamente, deve ser originada indiretamente usando outras medidas diretas. Métricas orientadas a função foram inicialmente propostas por Albrecht (1979), que sugeriu uma medida chamada ponto de função. Pontos por função são originados usando uma relação empírica baseada em medidas de contagem (direta) do domínio de informação do software e a avaliação da complexidade do software (PRESSMAN, 2001).

Ponto de função é uma unidade de medida de software, assim como, a hora é uma unidade de medida para o tempo. Para análise por ponto de função, os sistemas são divididos em cinco grandes classes e características gerais de sistema. As três primeiras classes são entradas externas, saídas externas e requisição externa conhecidas como transações. As duas próximas são arquivos lógicos internos e arquivos de interface externa, onde os dados são armazenados. Essas características avaliam a funcionalidade geral do sistema.

Análise de pontos de função foi desenvolvido com o objetivo de suprir dificuldades apresentadas pela linha de código, como medida de tamanho de software, e de ajudar no desenvolvimento de um mecanismo que pudesse prever o esforço associado ao desenvolvimento de software (LONGSTREET, 2004).

Em 1984 o método foi refinado e, posteriormente, com o aumento do uso da análise por pontos de função, tornou-se necessário um guia que interpretasse as regras originais para os novos ambientes. Devido a essa necessidade em 1986 foi criado o "International Function Point Group" (IFPUG). A partir desta data, o IFPUG passou a ser o responsável pela definição das regras de contagem, pelo treinamento e pela certificação dos profissionais interessados na aplicação dessa métrica e divulgação de diversas bases de dados históricas de produtividade da indústria de desenvolvimento de software, disponibilizadas por vários órgãos, entre eles o "International Software Benchmarking Standards Group". Essas informações possibilitam a estimativa de tempo de desenvolvimento de novos projetos de software e produtividade da equipe com base em estimativas anteriores realizadas através da análise por pontos de função (GARMUS; HERRON, 2000) (IFPUG, 2000) (LONGSTREET, 2004).

Essa métrica tem como princípio básico focar na especialização de requisitos para obter estimativas de tempo, de custo, de esforço e de recursos na fase inicial do projeto ou na manutenção de software independente da tecnologia utilizada para implementação (FUREY, 1997) (RIBU, 2001).

A análise por ponto de função permite uma contagem indicativa no início do desenvolvimento sem conhecer detalhes do modelo de dados. Posteriormente, na fase de projeto, essa contagem passa a ser uma estimativa com maior precisão da complexidade das funções e, ao término do desenvolvimento do software, na implantação, é realizada uma contagem detalhada, obtida a partir do grau de complexidade das funções levantadas no processo funcional, modelo de dados, descrição das telas, relatórios (IFPUG, 2000) (LONGSTREET, 2004).

Em geral, as organizações aplicam a análise por ponto de função como: um método para determinar o tamanho do pacote de software adquirido; para apoiar a análise da produtividade e qualidade; para estimar os custos, recursos e esforços de projetos de desenvolvimento e manutenção de software (GARMUS; HERRON, 2000) (HAZAN, 1999).

Para estimar o tamanho do software de acordo com a análise por ponto de função, o IFPUG definiu os procedimentos de contagem. O processo de contagem compõe-se de sete etapas (IFPUG, 2000):

1. **Determinar o tipo de contagem:** a análise por ponto de função se propõe a estimar o tamanho de projetos de desenvolvimento; aplicações em uso ou projetos de manutenção.
2. **Identificar o escopo da contagem e a fronteira da aplicação:** o escopo define as funções que serão contadas de acordo com a visão do usuário, e a fronteira da aplicação separa o projeto a ser contado dos usuários e das aplicações externas ao domínio do projeto.
3. **Contar as funções de dados:** as funções de dados consistem de: arquivos lógicos internos (ALIs) e arquivos de interface externa (AIEs). Um arquivo lógico interno de uma aplicação sempre será contado como um ALI na aplicação de origem. Para identificar um arquivo lógico interno, observar as seguintes regras:
    - Se o grupo de dados ou informações de controle é lógico e identificável pelo usuário;
    - Se o grupo de dados é mantido através de um processo elementar dentro da fronteira da aplicação a ser contada.

A complexidade dos arquivos lógicos internos é identificada através do número de itens de dados e de registros lógicos. Para identificar os itens de dados, seguir as regras:

- Contar um item de dados para cada campo reconhecido pelo usuário como único, não repetido, mantido por um arquivo lógico interno ou recuperado de um arquivo de interface externa ou de um arquivo lógico interno.
- Quando duas aplicações mantêm e ou referenciam o mesmo arquivo lógico interno ou arquivo de interface externa, mas cada uma mantêm ou referencia itens de dados separados, conte apenas os itens de dados utilizados em cada aplicação para avaliar a complexidade dos arquivos lógicos internos e arquivos de interface externa.
- Contar um item de dado para cada campo requerido pelo usuário para estabelecer um relacionamento com outro arquivo lógico interno ou arquivo de interface externa que, no caso, é considerada chave estrangeira.

Para identificar um arquivo de interface externa, são verificados grupos de dados ou de informação de controle que satisfaçam a definição de um arquivo de interface externa. As seguintes regras devem ser verificadas:

- O grupo de dados ou informação de controle é lógico e identificável pelo usuário;
- O grupo de dados é referenciado pela aplicação a ser contada, mas pertence a outra aplicação fora da fronteira da aplicação;
- O grupo de dados não mantido pela aplicação a ser contada;
- O grupo de dados é mantido em um Arquivo Lógico Interno de outra aplicação.

De acordo com o número de itens de dados e de registros lógicos identificados, classifica-se o arquivo lógico interno e o arquivo de interface externa conforme a tabela 3.1 e são atribuídos os pesos de 7, 10 ou 15 pontos de função para os arquivos lógicos internos e 5, 7 ou 10 pontos de função para os arquivos de interface externa respectivamente, à complexidade baixa, média ou alta.

Registros lógicos	Itens de dados referenciados		
	1 a 19	20 a 50	51 ou mais
1	baixa	baixa	média
2 a 5	baixa	média	alta
6 ou mais	média	alta	alta

Tabela 3.1 – Complexidade funcional

(IFPUG, 2000)

4. **Contar as funções transacionais:** estas funções representam as funções de processamento dos dados fornecidos pela aplicação ao usuário. As funções transacionais podem ser entradas externas (EE), saídas externas (SE) e consultas externas (CE). Compõe-se de itens referenciados (parâmetros que estão sendo representados pelos seus tipos) e de arquivos referenciados.

Para identificar as entradas externas, devem ser analisados todos os processos elementares que recebem dados de fora da aplicação e que atualizam um ou mais arquivo lógico interno de acordo com as seguintes regras:

- Os dados ou a informação de controle devem ser recebidos de fora da fronteira da aplicação;
- No mínimo, um arquivo de interface externa é mantido, se a entrada de dados pela fronteira não for informação de controle que altera o comportamento do sistema.

Para contabilizar o item de dado referenciado na transação de EE, considerar as seguintes regras:

- Os itens de dados de acordo com a visão do usuário;

- Contar os itens de dados, que aparecem mais de uma vez em um arquivo, por causa da tecnologia ou da técnica de implementação, apenas uma vez;
- Considerar um item de dado adicional para as teclas de função ou linha(s) de comando que especificam a ação a ser tomada pela entrada externa;
- Contabilizar os campos não informados pelo usuário, mas que são atualizados em um arquivo lógico interno, por uma entrada externa;
- As mensagens de erros, confirmação ou itens de dados adicionais, devem ser considerados, caso sejam requeridas pelo usuário.

De acordo com o número de itens de dados, arquivo lógico interno e arquivo de interface externa referenciados, classifica-se a entrada externa em baixa, média ou alta, conforme a tabela 3.2 abaixo e os respectivos pesos (3, 4 ou 6 pontos de função).

Para identificar as saídas externas, verificar o processamento lógico do processo elementar de acordo com as seguintes regras:

Arquivos referenciados	**Itens de dados referenciados**		
	1 a 4	5 a 15	16 ou mais
0 ou 1	baixa	baixa	média
2	baixa	média	alta
3 ou mais	média	alta	alta

Tabela 3.2 – Complexidade de entrada externa

(IFPUG, 2000)

- se contêm no mínimo, uma fórmula matemática ou cálculo;
- se cria dados derivados;
- se mantêm no mínimo, um arquivo lógico interno;
- se altera o comportamento do sistema.

Já a complexidade da saída externa é verificada através do número de arquivos e elementos de dados referenciados. Deve-se contar um arquivo referenciado para cada arquivo lógico interno mantido ou lido ou um arquivo de interface externa lido durante o processamento da saída externa.

Para identificar os itens de dados, observar as regras:

- Contar um item de dado para cada campo reconhecido pelo usuário, não repetido, que entre pela fronteira da aplicação e seja requerido para especificar quando, qual e/ou como o dado será recuperado ou gerado pelo processo elementar ou que saia da fronteira da aplicação. Se um item de dado entra e sai pela fronteira da aplicação, contá-lo apenas uma vez;
- Contar um item de dado quando a aplicação enviar mensagens de resposta para fora da fronteira, indicar um erro ocorrido durante o processamento, confirmar o processamento completo ou verificar se o processamento deve continuar;
- Contar um item de dado para a habilidade de especificar uma ação a ser tomada, quando existem múltiplos métodos para chamar o mesmo processo lógico;
- Não contar os campos que são recuperados ou derivados pelo sistema e armazenado sem um arquivo lógico interno durante o processo elementar se esses campos não cruzam a fronteira da aplicação;
- Não contar variáveis, números de páginas, data/hora ou selos gerados pelo sistema.

Após identificar os arquivos e os itens de dados referenciados, classificar a saída externa (SE) conforme a tabela 3.3 abaixo e atribuir o peso 4, 5 ou 7 pontos de função respectivo à complexidade baixa, média ou alta.

Arquivos referenciados	Itens de dados referenciados		
	1 a 5	6 a 19	20 ou mais
0 ou 1	baixa	baixa	média
2 ou 3	baixa	média	alta
4 ou mais	média	alta	alta

Tabela 3.3 – Complexidade de saída externa

(IFPUG, 2000)

A consulta externa visa apresentar informação para o usuário através da recuperação de dados ou informação de controle. Verificar se o processamento lógico é elementar, segundo as regras abaixo:

- Recupera dados ou informação de controle de um arquivo lógico interno ou arquivo de interface externa;
- Não contêm fórmulas matemáticas ou cálculos;
- Não cria dados derivados;
- Não mantêm arquivo lógico interno;
- Não altera o comportamento do sistema.

De acordo com o número de itens de dados e arquivos referenciados, classificam-se as consultas externas conforme a tabela 3.4 e atribui-se o peso 3, 4 ou 6 pontos de função conforme a complexidade baixa, média ou alta.

5. **Determinar o ponto de função não ajustado:** após identificar as funções de dados e transacionais, multiplicar o total de arquivo lógico interno, arquivo de interface externa, entrada externa, saída externa e consulta externa pela respectiva complexidade para determinar o valor de ponto de função não ajustado. De acordo com a complexidade, cada uma das funções de dados e transacionais contribui com um determinado número de ponto de função. O

total desses pontos de função computados são chamados de pontos de função não-ajustados.

Arquivos referenciados	Itens de dados referenciados		
	1 a 5	6 a 19	20 ou mais
0 ou 1	baixa	baixa	média
2 ou 3	baixa	média	alta
4 ou mais	média	alta	alta

Tabela 3.4 – Complexidade de consulta externa

(IFPUG, 2000)

6. **Determinar o fator de ajuste:** o nível de influência dos fatores de ajustes técnicos é determinado com base nas 14 características gerais de sistemas, conforme mostra a tabela 3.5. Após atribuir e somar os valores dos fatores de ajustes para obter o nível de influência da aplicação, deve-se aplicar a seguinte fórmula:

FATOR DE AJUSTE = (Nível de influência * 0,01) + 0,65

Características gerais do sistema	
Comunicação de dados	Atualização on-line
Processamento distribuído	Processamento complexo
Desempenho	Reutilização de código
Utilização de equipamentos	Facilidade de implantação
Volume de transações	Facilidade operacional
Entrada de dados on-line	Múltiplos locais
Eficiência do usuário final	Facilidade de mudanças

Tabela 3.5 – Características gerais do sistema

(IFPUG, 2000)

Estas características influenciarão a complexidade do software e conseqüentemente seu tamanho. As características gerais do software podem variar o tamanho do software em torno de 35%.

7. **Calcular os pontos de função ajustados:** os pontos de função ajustados são calculados a partir dos pontos de função não ajustados e do fator de ajuste de acordo com a seguinte fórmula:

   PF_DESENVOLVIMENTO=PF_NAOAJUSTADO*FATOR_AJUSTE (IFPUG, 2000).

A análise por ponto de função tem muita utilidade, porém existem restrições quanto a medição de esforço em manutenção (correção de defeitos). Muito do esforço associado à correção de defeitos está ligado a tentar resolver e entender o problema. Outro problema inerente a medir o esforço em manutenção é que muito desse trabalho é feito por uma ou duas pessoas. (LONGSTREET CONSULTING INC, 2003).

3.4 Ponto por Caso de Uso

O ponto por caso de uso (PCU) é um método de estimativa de tamanho de projeto de software orientado a objetos criado por Karner (1993), com base na análise por ponto de função e no modelo de casos de uso. O modelo de casos de uso foi posteriormente incorporado na "Unified Modeling Language" (UML) e tem sido muito utilizado no mercado atualmente. O ponto por caso de uso também estima o tamanho da função do software de acordo com a visão do usuário final e com base em uma unidade de medida o PCU.

Neste método, Karner (1993) explora o modelo e a descrição de casos de uso, substitui algumas características técnicas propostas pela análise por ponto de função, cria os fatores ambientais e propõe uma estimativa de produtividade.

Karner (1993) propôs a produtividade de 20 homens/hora por ponto por caso de uso. Com base nos recursos utilizados pelos projetos de seu estudo, tentou ainda encontrar a correlação entre ponto por caso de uso e os recursos necessários para desenvolver o projeto através da adoção da regressão linear, mas seus dados não foram suficientes para encontrar a relação proposta.

O método ponto por caso de uso contribui para a diminuição de algumas dificuldades impostas pelo mercado em relação à resistência de adoção de métodos de estimativas, porque é um método simples, fácil de usar e rápido de se aplicar, quando possui as informações necessárias para realizar as estimativas (DAMODARAN; WASHINGTON, 2002).

O ponto por caso de uso também possui um processo de contagem conforme descrito a seguir:

- **Contar os atores e atribuir o grau de complexidade:** identificar e multiplicar o total de atores de acordo com o tipo de complexidade (simples, médio ou complexo) pelo respectivo peso (1, 2 ou 3), conforme tabela 3.6 e somar os produtos para obter o total de atores não ajustados.

Complexidade do ator	Descrição	Peso
Simples	Representa um outro sistema com interface definida de programa	1
Médio	Representa um outro sistema que interage através de protocolos ou quando há interação humana através de terminal	2
Complexo	É uma pessoa que interage através de interface gráfica ou Web	3

Tabela 3.6 –Peso dos atores

(KARNER, 1993)

- **Contar os casos de uso e atribuir a complexidade:** a complexidade é baseada no número de transações. De acordo com o número de transações, multiplicar cada caso de uso pelo respectivo peso conforme tabela 3.7.

Complexidade do caso de uso	Descrição	Peso
Simples	Tem até 3 transações incluindo os passos alternativos e deve ser realizado com menos de 5 classes de análise	5
Médio	Tem de 4 a 7 transações incluindo os passos alternativos e deve ser realizado com 5 a 10 classes de análise	10
Complexo	Tem acima de 7 transações incluindo os passos alternativos e deve ser realizado com pelo menos de 10 classes de análise	15

Tabela 3.7 – Peso dos casos de uso

(KARNER, 1993)

- **Calcular os PCU não ajustados:** de acordo com a seguinte fórmula:
  PCU não ajustados = Σ peso de atores + Σ pesos de casos de uso.
- **Determinar o fator de complexidade técnica (tabela 3.8):** os fatores de complexidade técnica variam numa escala de 0 a 5, de acordo com o grau de dificuldade do sistema a ser construído. O valor 0 indica pouca criticidade e baixa complexidade (irrelevante para o projeto) e o valor 5 indica alta criticidade e complexidade (essencial). Após determinar o valor dos fatores, multiplicar pelo respectivo peso, somar o total e aplicar a seguinte fórmula:
  Fator de complexidade técnica = 0,6 + (0,01*Somatório do fator técnico).

Descrição	Peso
Sistemas distribuídos	2
Desempenho da aplicação	1
Eficiência do usuário final	1
Processamento interno complexo	1
Reusabilidade do código em outras aplicações	1
Facilidade de instalação	0,5
Usabilidade	0,5
Portabilidade	2
Facilidade de alteração	1
Concorrência	1
Características especiais de segurança	1
Acesso direto para terceiros	1
Facilidades especiais de treinamento	1

Tabela 3.8 – Fatores de complexidade técnica
(KARNER, 1993)

- **Determinar a eficiência do fator ambiental:** os fatores ambientais indicam a eficiência do projeto e estão relacionados ao nível de experiência dos profissionais. Esses fatores (Tabela 3.9) são determinados através da escala de 0 a 5, onde 0 indica baixa habilidade (pouca experiência no domínio da aplicação), 3 indica média experiência e 5 indica alta experiência. Por exemplo, para o fator motivação, 0 significa sem motivação para o projeto; 3, média motivação e 5, alta motivação. Para o fator requisitos estáveis, 0 significa extremamente instável; 3, médio e 5, estável. Para o fator trabalhadores com dedicação parcial, 0 indica poucos ou nenhum profissional de período não integral; 3, médio e 5 todos profissionais de período não integral. Para o fator dificuldade da linguagem de programação, 0 indica que a linguagem é de pouca complexidade e ou que a equipe de projeto tem domínio completo sobre a mesma; 3, médio e 5, muita dificuldade com a linguagem de programação. Após determinar o valor de cada fator, multiplicar pelo peso e somar o total dos valores. Em seguida, aplicar a seguinte fórmula:
  Fator de complexidade ambiental = 1,4 + (-0,03 * somatório do fator ambiental)

Descrição	peso
Familiaridade com o processo de desenvolvimento de software	1,5
Experiência na aplicação	0,5
Experiência com OO, na linguagem e na técnica de desenvolvimento.	1,0
Capacidade do líder do projeto	0,5
Motivação	1,0
Requisitos estáveis	2,0
Trabalhadores com dedicação parcial	-1,0
Dificuldade da linguagem de programação	-1,0

Tabela 3.9 – Fatores de complexidade ambiental

(KARNER, 1993)

- **Calcular os pontos de caso de uso ajustados:** esse cálculo é realizado com base na multiplicação dos pontos de casos de uso não ajustados, na complexidade técnica e na complexidade ambiental através da fórmula:
  PCU = PCU não ajustados * fator de complexidade técnica * fator de complexidade ambiental (KARNER, 1993).

3.5 Considerações Finais

Damodaran e Washington (2002) acreditam que não é difícil para uma organização desenvolver seus próprios padrões de granularidade dos casos de uso de análise e classificá-los de acordo com a complexidade. Se a organização tiver padrões para definir casos de uso, é possível ter uma boa base de dados histórica de estimativas para ser utilizada como fonte de consulta para estimar novos projetos. Estes autores acreditam que o ponto por caso de uso tem potencial para se tornar um método maduro e largamente aceito como um método de estimativa.

O ponto por caso de uso é baseado no modelo de casos de uso, muito utilizado atualmente no mercado para identificar e documentar os requisitos do software nos termos dos usuários e de maneira mais completa e articulada que outros métodos (DEKKERS, 1999).

Conforme apresentado, existem restrições quanto a aplicação de pontos de função para o cálculo de esforço em manutenção. Portanto, a proposta é utilizar ponto de caso de uso como a métrica, por estar associada ao caso de uso, que é o artefato escolhido para documentação de requisitos.

4. PROCESSO DE MANUTENÇÃO ATUAL DA EMPRESA

4.1 Introdução

O objetivo desse capítulo é apresentar o processo de manutenção de software atual da empresa escolhida como modelo. Portanto, são apresentados o fluxo de trabalho, os artefatos utilizados, os papéis de cada profissional e as ferramentas utilizadas.

Outro objetivo é estudar um processo de manutenção que esteja em utilização e que possa ser avaliado, provendo subsídios, para a elaboração de uma proposta de melhoria do processo de manutenção de software.

4.2 Fluxo de Trabalho do Processo de Manutenção

A empresa possui um departamento de informática, sendo dividido em duas áreas:

- **Helpdesk**: responsável pelo atendimento aos usuários com dúvidas referente à utilização de sistemas, e responsável por encaminhar solicitações de manutenção para a área de desenvolvimento de sistemas através de chamado.
- **Desenvolvimento de Software**: responsável pelo atendimento das solicitações encaminhadas pelo helpdesk.

No fluxo do processo por chamado, apresenta-se como a solicitação de uma manutenção é atendida, do seu início até a solução.

O processo inicia-se através da solicitação de atendimento que é feita por e-mail ou por telefone. Os chamados são abertos no Fireman (FIREMAN ENTERPRISE, 2003), sistema onde são registrados e gerenciados os chamados dos usuários pela área de informática, são analisados em relação à especificação, ao tempo de atendimento e

à sua prioridade. É implementado de acordo com a especificação, testado e implantado no ambiente de produção onde o chamado é homologado, conforme apresentado na fig. 4.1.

A representação dos fluxos estão apresentados conforme referência (FIPS PUBS IDEF0,1993).

Caso o chamado não seja homologado, por não estar de acordo com a solicitação, é aberto outro chamado vinculado ao original, classificado como retorno de homologação. Todas as fases do fluxo do chamado são realizadas novamente até que o atendimento realizado esteja de acordo com a solicitação feita.

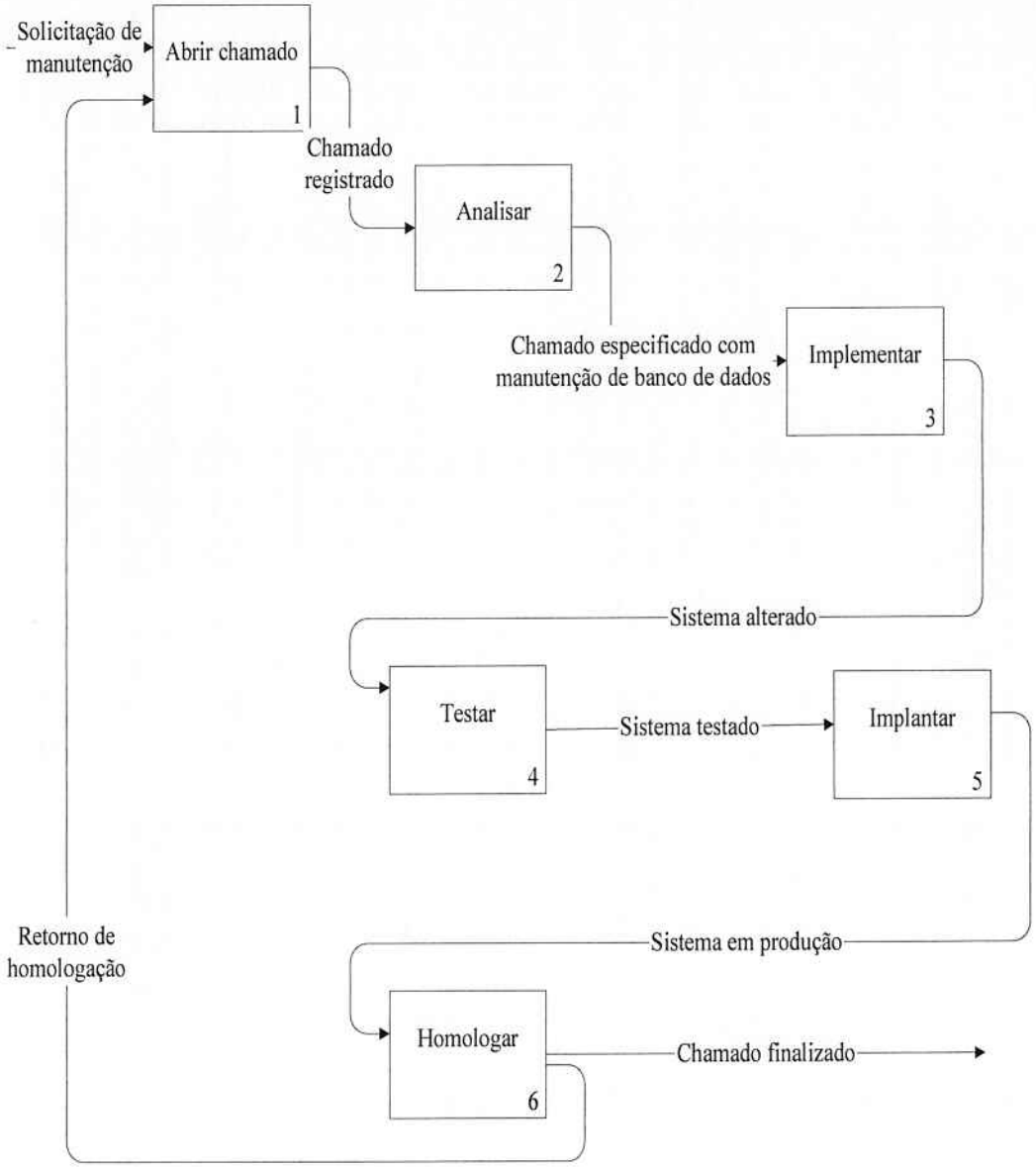

Figura 4.1 – Processo para o Fluxo de Chamados por função

O usuário inicia o processo através da solicitação de atendimento que é feita por e-mail ou por telefone.

A área de helpdesk, responsável em receber as solicitações e registrar os chamados no Fireman como apresentado no exemplo da fig.4.2, é integrada à área de

desenvolvimento de sistemas. E é a mesma que envia o chamado para o coordenador de desenvolvimento de sistemas, que faz uma análise preliminar, estabelecendo tempo, prioridade conforme a tabela 4.1, viabilidade e classificação dos chamados conforme a tabela 4.2.

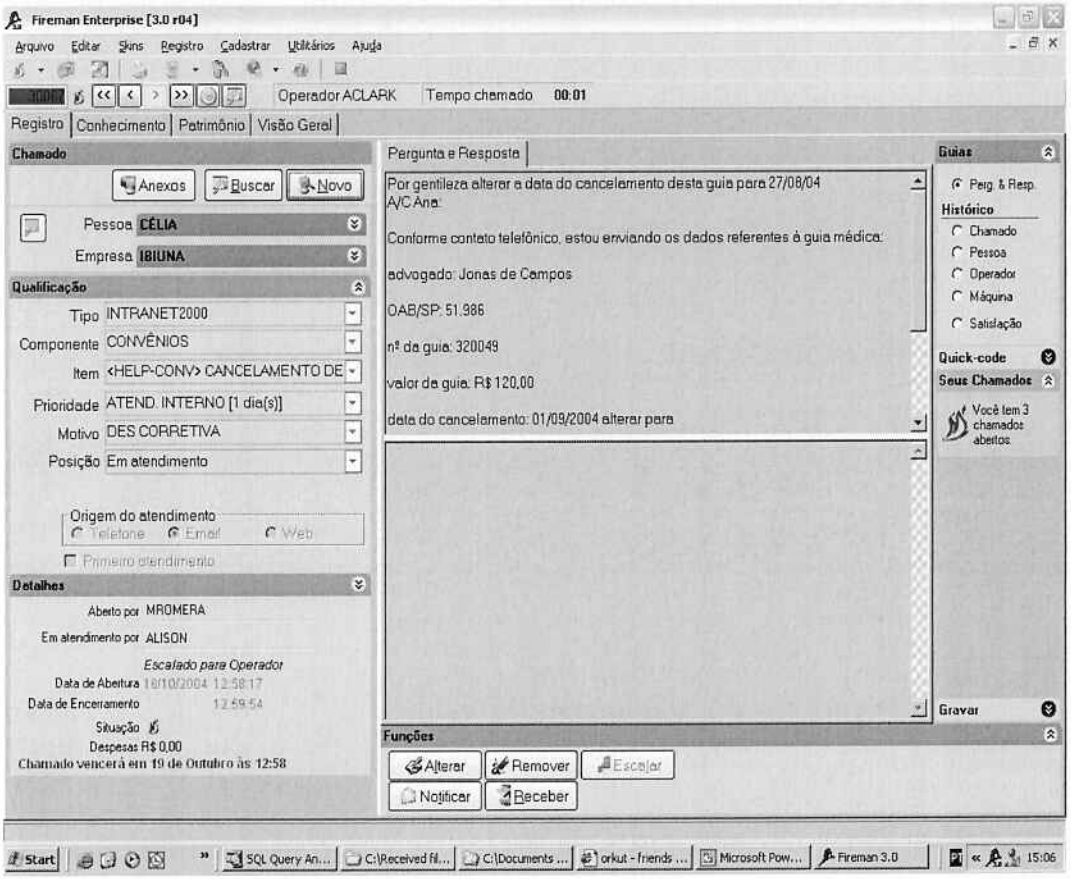

Figura 4.2 – Um Exemplo de Aplicação do Fireman

Descrição	Prazo	Procedimentos
Urgente	No mesmo dia	Se estiver impactando na operação do sistema, deve ser parada qualquer atividade que o analista estiver desenvolvendo no momento, que não seja da mesma prioridade.
Rápido	3 dias	A operação do sistema não está sendo impactado, mas necessita de um atendimento rápido dentro do prazo máximo estabelecido.
Curto Prazo	5 a 10 dias	Não impacta na operação do sistema, mas deve ser atendido em um curto prazo máximo estabelecido.
Médio Prazo	15/30/45/60 dias	Não impacta na operação do sistema, normalmente são manutenções evolutivas que podem esperar um tempo considerado de médio prazo para atendimento.
Longo Prazo	Mais de 60 dias	Não impacta na operação do sistema, normalmente são manutenções evolutivas que podem esperar um tempo considerado de longo prazo para atendimento.

Tabela 4.1 - Prioridade de chamados

Descrição	Aplicação	Exemplo
Corretiva	Erro encontrado no sistema que deve ser corrigido dentro da prioridade estabelecida.	O sistema não grava a alteração de um produto.
Evolutiva	Melhorias detectadas pelos usuários ou pelo pessoal de desenvolvimento de sistemas a fim de melhorar o sistema.	Integração de coletor de dados para efetuar o balanço de estoque em que é feito direto pelo leitor de código de barras.
Atualização de banco de dados	Atualizações e inclusões no banco de dados em que a interface do sistema não permite ao usuário ter independência da área de sistemas.	Alteração do número de uma nota fiscal já conferida e dada entrada no estoque sendo que o sistema não tem a opção de alterar dados da nota fiscal após a mesma ser conferida pelo usuário
Consulta de banco de dados	Extração de informações do banco de dados em formato de relatório que o sistema não oferece e que não é necessário com freqüência.	Relatório de produtos que estão com os preços diferenciados de região para região.

Tabela 4.2 - Classificação de chamados

O chamado é enviado ao administrador de banco de dados para manutenção quando necessário, que repassa para o coordenador de desenvolvimento de sistemas seguir o fluxo.

O analista programador recebe o chamado, atende, testa, implanta e envia o chamado para o homologador solicitar que o usuário teste e homologue a solicitação feita, para que o homologador encerre o chamado, conforme apresentado na fig. 4.3.

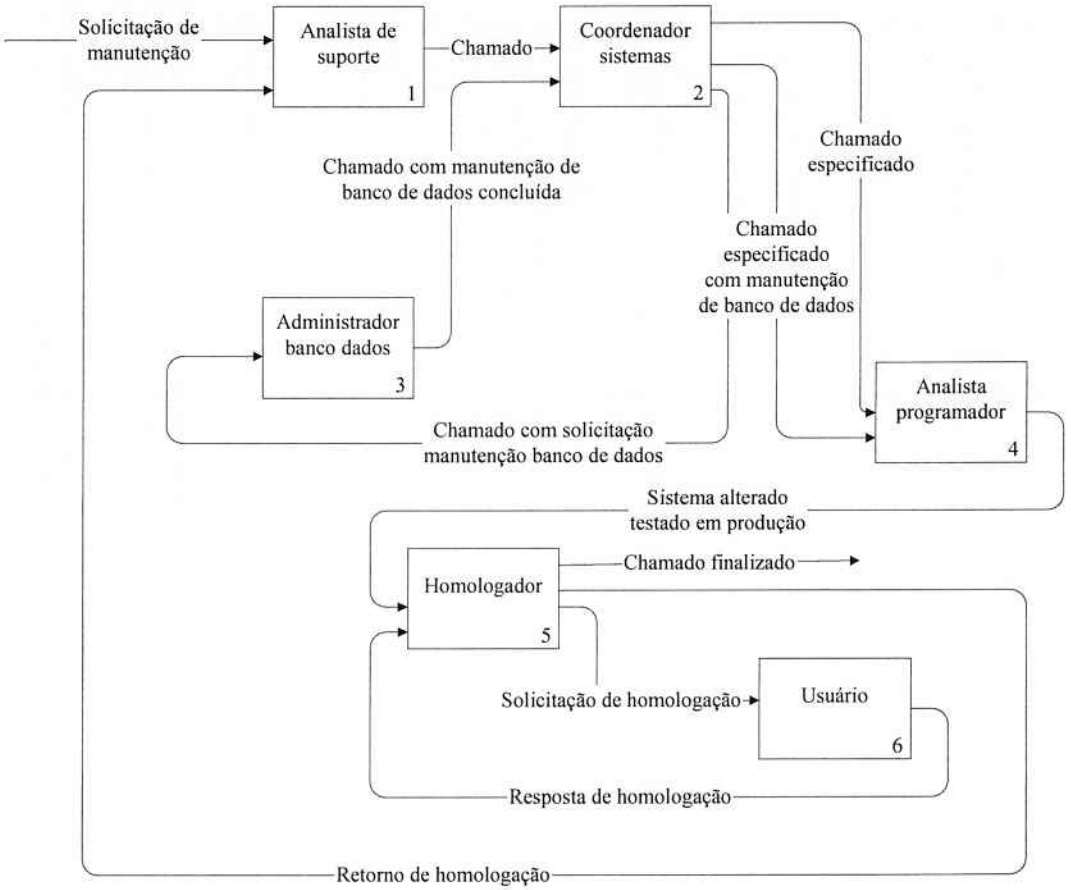

Figura 4.3 – Processo para o Fluxo de Chamados por profissional

4.3 Papéis no Processo

A empresa possui diversos papéis no processo, sendo que algumas vezes as atividades são executadas pela mesma pessoa, dependendo da demanda de trabalho, abaixo segue os papéis utilizados no processo:

- **Analista de suporte**: é responsável por receber as solicitações dos usuários, registrar o chamado e enviar para o coordenador de desenvolvimento de sistemas continuar o fluxo do processo de manutenção.
- **Coordenador de desenvolvimento de sistemas**: é responsável por receber as solicitações de usuários através do analista de suporte. O coordenador analisa o chamado, devolve para o analista de suporte quando não estiver de acordo ou quando for detectado que não é de desenvolvimento de sistemas, envia o chamado para o analista programador que desenvolve a solução. Após estas fases o coordenador acompanha os chamados até o encerramento do mesmo pelo analista programador.
- **Analista programador**: recebe o chamado do coordenador de desenvolvimento de sistemas, verifica a prioridade de desenvolvimento e atende os chamados por prioridade, acompanhando o prazo estipulado. É responsável por instalar as alterações necessárias em produção de acordo com as especificações definidas pelo coordenador, na solicitação do usuário, testado e funcionando para que o usuário apenas confirme que a solicitação está homologada.
- **Homologador**: responsável por receber chamados dos analistas programadores e do coordenador de desenvolvimento para acompanhar junto ao usuário se a solicitação feita pelo mesmo está de acordo. Para confirmar o atendimento, o homologador recebe um e-mail de confirmação do usuário, e anexa uma cópia deste e-mail ao chamado, caso contrário o homologador recebe um e-mail indicando que a solicitação não está de acordo e o mesmo é anexado ao chamado. É aberto outro chamado vinculado ao original, com a

classificação de retorno de homologação para gerenciamento e controle de qualidade dos chamados pelo coordenador de desenvolvimento de sistemas.

- **Usuário**: é responsável pelas solicitações de manutenção de sistemas e de verificar se o que foi desenvolvido está de acordo com o que foi solicitado, homologando o chamado, permitindo encerrar a solicitação.
- **Administrador de banco de dados**: é responsável por qualquer manutenção no banco de dados, mediante as necessidades identificadas pelo analista programador ou pelo coordenador de desenvolvimento de sistemas e por atualizações no banco de dados de produção, que são premissas para implantação de qualquer alteração em produção.

4.4 Artefatos utilizados

O modelo de banco de dados é sempre atualizado em qualquer manutenção e é um artefato obrigatório para o processo atual.

A ferramenta de controle de versão também é muito utilizada, permitindo que as alterações realizadas sejam documentadas, incluindo o responsável pela alteração evitando possíveis falhas de desenvolvimento.

4.5 Ferramenta utilizada

A ferramenta utilizada para gerenciamento da fase de manutenção de software é o Fireman, líder nacional em software para helpdesk, existente há mais de 8 anos no mercado nacional e utilizado em mais de cem empresas em todo o país (SIAL SOFTWARE, 2004).

Fireman é um aplicativo 100% nacional para registro de chamados, suporte técnico, utilizando tecnologia de três camadas, pode ser utilizado tanto para suporte interno quanto externo, integrando-se aos bancos de dados já existentes.

Permite o gerenciamento de solicitações de serviços, contendo as seguintes características destacadas, que são utilizadas no processo atual:

- **Registro de chamados:** chamados são mantidos em banco de dados, possibilitando um acompanhamento das fases de execução de serviço pela área técnica;
- **Escalação:** chamados podem ser transferidos para outros técnicos ou grupos de solucionadores conforme suas especialidades, com acompanhamento total de passos, histórico de registros, datas, horários e consumo de tempo;
- **Pesquisa de satisfação:** e-mails automáticos podem ser disparados no encerramento do chamado, solicitando manifestação do usuário quanto ao atendimento;
- **Gráficos e relatórios:** todos os gráficos são gerados dentro do MS Excel, permitindo grande leque de formatações para o documento final. Os vários relatórios existentes podem ser exportados para diversos formatos (SIAL SOFTWARE; 2004).

4.6 Avaliação do processo de manutenção

O processo de manutenção utilizado na empresa, apresenta diversos pontos fracos a serem melhorados. Esses pontos são destacados tanto em relação à ferramenta quanto em relação ao processo, como apresentado a seguir:

- **Ferramenta:**
    - Não possui o tempo orçado para manutenção, o que impacta na qualidade de software, ou seja, fica impossível estimar o prazo para

entrega da alteração no sistema para o usuário sem o tempo orçado de cada atividade;

- Não apresenta o apontamento detalhado das horas utilizadas em cada atividade dentro do chamado, impossibilitando o gerenciamento de tempo de cada atividade;
- Não apresenta uma função de priorização adequada de chamados, tornando difícil à visualização de que atividade o analista programador está desenvolvendo no momento;
- Não apresenta uma consulta de posição dos chamados pendentes;

- **Processo:**
  - Falta de formalização no processo quanto aos artefatos, que nem sempre estão disponíveis;
  - Falta de transparência ao usuário do que está sendo feito, e em que prazo. Muitas vezes a solicitação do usuário é de prioridade mais baixa e fica pendente para desenvolvimento, e o usuário fica sem um retorno adequado de quando será realizado, muito menos quando estará pronta a solicitação, pois o processo não tem uma comunicação formal e a ferramenta não disponibiliza a consulta de chamados pendentes;
  - A falta de priorização adequada de chamados corporativos por um comitê de informática.

E sobre os pontos fracos podemos citar algumas melhorias para a ferramenta e para o processo de manutenção:

- **Ferramenta:**
  - Criar o campo para preenchimento do tempo estimado de cada atividade;
  - Criar uma função de apontamento de horas de cada analista programador relacionada com o chamado enviado e uma consulta onde

podem ser visualizados o tempo estimado e o tempo realizado de cada atividade e chamado;

- Criar uma função de priorização de chamados, onde o coordenador e o comitê de informática possam organizar as atividades por ordem de prioridade;
- Criar consulta para os usuários, supervisores e gerentes das áreas de negócios, acompanharem o estado de cada chamado efetuado pelas respectivas áreas.

- **Processo:**
  - Formalização do processo, estabelecendo artefatos mínimos para manutenção de sistemas;
  - Divulgar a consulta do estado do chamado por área, com intuito de formalizar a comunicação com o usuário e a cada alteração na priorização, ou na alteração do prazo estimado, avisar automaticamente por e-mail o usuário solicitante da manutenção do sistema, do que ocorreu, melhorando a transparência e antecipando a resolução de possíveis problemas;
  - Criação de um comitê de informática, composto por pessoas das áreas de negócio da empresa, que tenha autonomia para priorizar o que será feito e em que ordem de acordo com a capacidade de produção do departamento de informática da empresa.

5. PROPOSTA DE ALTERAÇÃO DO PROCESSO DE MANUTENÇÃO

5.1 Introdução

O objetivo desse capítulo é apresentar uma proposta de um processo de manutenção de software, que utiliza engenharia reversa, artefatos da UML e utilização de métrica de software, visando melhoria de qualidade de software e gerenciamento de tempo de projeto.

A proposta envolve o uso de engenharia reversa para a geração de artefatos UML. Como os projetos da empresa selecionada como modelo, foram desenvolvidos utilizando componentes, classes e casos de uso, atualmente desatualizados, a proposta inclui a utilização de artefatos da UML, diagramas de classes, diagramas de caso de uso e casos de uso, que são obtidos através de engenharia reversa, viabilizando o processo de manutenção.

A proposta também utiliza engenharia reversa para obter o modelo entidade relacionamento do banco de dados através de ferramenta específica.

Para melhorar a qualidade de software e gerenciamento de tempo de projeto foi incluído na proposta a utilização de métrica de software, especificamente, ponto por caso de uso.

5.2. Modelo do processo

Baseado na avaliação do processo atual, nos pontos fracos identificados e nas melhorias sugeridas, é proposto um fluxo de processo de manutenção de software.

O fluxo do processo por chamado inicia-se através da solicitação de atendimento que é feita por e-mail ou por telefone.

Os chamados são abertos no Fireman, são analisados em relação à especificação e classificados conforme tabela 5.1.

Descrição	Aplicação	Exemplo
Corretiva	Erro encontrado no sistema que deve ser corrigido dentro da prioridade estabelecida.	O sistema não grava a alteração de um produto.
Evolutiva	Melhorias detectadas pelos usuários ou pelo pessoal de desenvolvimento de sistemas a fim de melhorar o sistema.	Integração de coletor de dados para efetuar o balanço de estoque em que é feito direto pelo leitor de código de barras.
Atualização de banco de dados	Atualizações e inclusões no banco de dados em que a interface do sistema não permite ao usuário ter independência da área de sistemas.	Alteração do número de uma nota fiscal já conferida e dada entrada no estoque sendo que o sistema não tem a opção de alterar dados da nota fiscal após a mesma ser conferida pelo usuário
Consulta de banco de dados	Extração de informações do banco de dados em formato de relatório que o sistema não oferece e que não é necessário com freqüência.	Relatório de produtos que estão com os preços diferenciados de região para região.

Tabela 5.1 - Classificação de chamados no processo proposto

Nesta proposta aplica-se a engenharia reversa, para obter os artefatos necessários à manutenção de software, já que no processo atual os artefatos estão desatualizados, utilização de métrica de estimativa de tamanho de software, ponto por caso de uso, já que o processo atual não permite estimar prazo para as solicitações.

Nesta proposta, o Fireman prioriza os chamados estimados para resolução máxima de 8 horas como rápido, conforme a tabela 5.2, e aprova estes chamados para serem atendidos pelo desenvolvimento de software.

Os demais chamados que são orçados com mais de 8 horas, o usuário preenche o campo de justificativa para manutenção do software, que é aprovada e classificada a prioridade conforme tabela 5.2 pelo comitê de informática.

Descrição	Prazo	Procedimentos
Urgente	No mesmo dia	Se estiver impactando na operação do sistema, deve ser parada qualquer atividade que o analista estiver desenvolvendo no momento, que não seja da mesma prioridade.
Rápido	3 dias	A operação do sistema não está sendo impactado, mas necessita de um atendimento rápido dentro do prazo máximo estabelecido ou a quantidade de horas orçadas para atendimento do chamado é de no máximo 8 horas.
Curto Prazo	5 a 10 dias	Não impacta na operação do sistema, mas deve ser atendido em um curto prazo máximo estabelecido.
Médio Prazo	15/30/45/60 dias	Não impacta na operação do sistema, normalmente são manutenções evolutivas que podem esperar um tempo considerado de médio prazo para atendimento.
Longo Prazo	Mais de 60 dias	Não impacta na operação do sistema, normalmente são manutenções evolutivas que podem esperar um tempo considerado de longo prazo para atendimento.

Tabela 5.2 - Prioridade de chamados no processo proposto

A proposta sugere outra melhoria que é o apontamento de horas utilizadas para desenvolvimento por atividade, o teste do serviço, a implantação no ambiente de produção onde o serviço é homologado, como é apresentado na fig. 5.1.

Caso o chamado não seja homologado por não estar de acordo com a solicitação, é aberto outro chamado vinculado ao original classificado como retorno de homologação. Todas as fases do fluxo do chamado são realizadas novamente até que o atendimento realizado esteja de acordo com a solicitação feita.

Nesta proposta uma melhoria importante é a formalização da transparência com o usuário, que desde o início do fluxo do chamado é possível consultar o chamado, para saber a posição atual e a previsão de atendimento do chamado pela área de desenvolvimento de sistemas, sendo avisado a cada alteração no prazo previsto para início e para atendimento do chamado, automaticamente por e-mail.

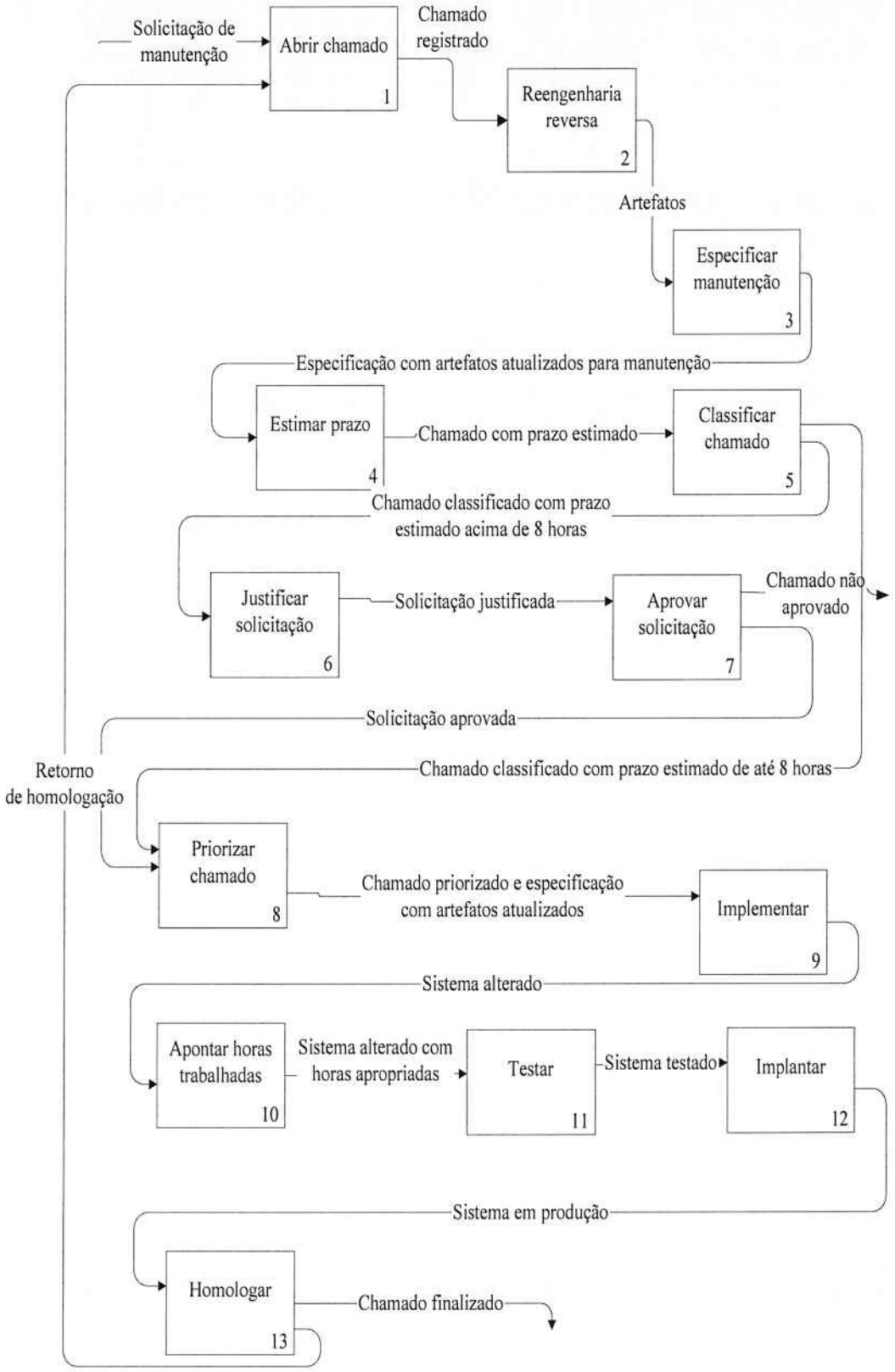

Figura 5.1 – Processo Proposto para o Fluxo de chamados por Função

O fluxo de chamados por profissional é iniciado pelo usuário através da área de helpdesk, responsável em receber as solicitações e registrar os chamados no Fireman.

O analista de suporte envia o chamado para o coordenador de desenvolvimento de sistemas, que faz uma análise preliminar e classifica os chamados conforme a tabela 5.1. Na proposta é adicionada a atividade de engenharia reversa para geração dos artefatos necessários para manutenção, é especificado a manutenção de sistema, é estimado o prazo para as atividades, utilizando métricas de softwares incluída na proposta para melhoria da qualidade de software.

Nesta proposta o usuário deve justificar a solicitação que está sendo feita, caso o prazo estimado ultrapasse 8 horas de trabalho, pois o comitê de informática avalia a solicitação, a justificativa, o custo, aprova a manutenção solicitada e prioriza os chamados de acordo com a tabela 5.2.

O coordenador de desenvolvimento de sistemas envia o chamado para o administrador de banco de dados para manutenção quando necessário, que devolve o chamado assim que efetivado a atividade.

O analista programador recebe o chamado, atende, testa e envia para o analista de qualidade que é responsável por testar e implantar as alterações em produção. O analista de qualidade após executar estas fases envia o chamado para o homologador que envia o chamado para o usuário que também tem o papel de testar a solicitação e homologar, como é apresentado na fig. 5.2.

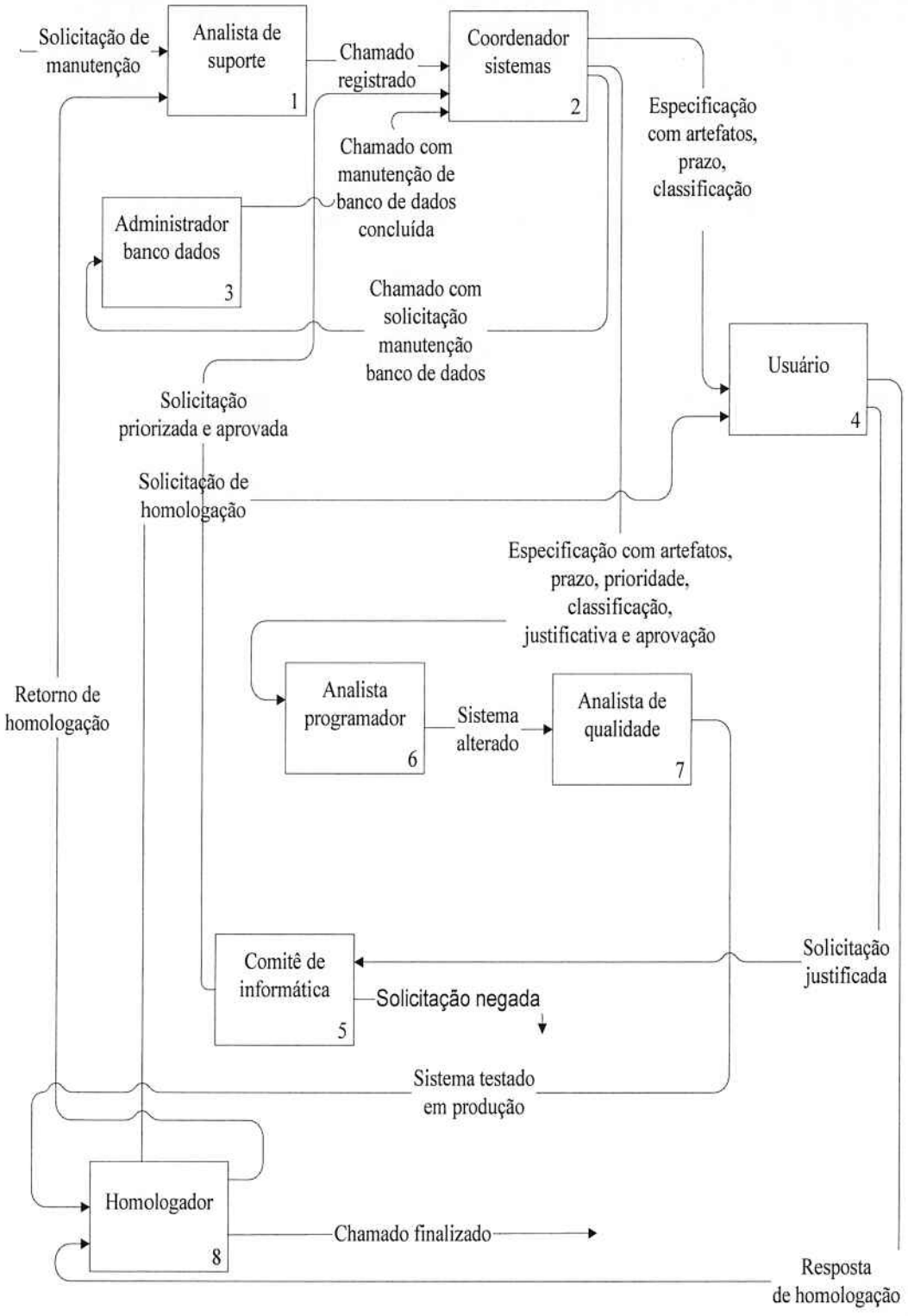

Figura 5.2 - Processo Proposto para o Fluxo de Chamados por profissional

5.3 Atividades do processo

A proposta envolve várias atividades dentro do processo, elas são executadas seqüencialmente, assim como no modelo cascata. Abaixo segue as atividades utilizadas no processo, conforme apresentado na fig. 5.1:

1. **Abrir chamado:** é a atividade de receber a solicitação de manutenção pelo usuário e atender pelo analista de suporte. A solicitação de manutenção é feita através do Fireman, gerando um chamado com um número, informando o usuário, que este pode consultar o chamado em qualquer fase de atendimento. O chamado após ser atendido pelo analista de suporte, é encaminhado para o coordenador de sistemas que o analisa de forma preliminar o chamado.
2. **Reengenharia reversa:** é a atividade de geração ou atualização dos artefatos propostos:
    - modelo de banco de dados através de ferramenta específica;
    - descrição dos casos de uso que devem ser escritos ou atualizados, de acordo com a manutenção, através da análise do código fonte e da aplicação;
    - diagramas de casos de uso que devem ser atualizados ou desenhados em relação à manutenção solicitada;
    - diagrama de classes que deve ser atualizado a cada manutenção a partir do código através de ferramenta específica.
3. **Especificar manutenção:** é a atividade de análise e projeto, sobre os artefatos gerados na reengenharia reversa. Nesta atividade, a saída é a especificação e os artefatos atualizados para atender a solicitação de manutenção;
4. **Estimar prazo:** é a atividade de utilizar a métrica de estimativa de tamanho de software selecionada, o ponto por caso de uso, sobre os artefatos atualizados para atender a solicitação de manutenção, mais precisamente, sobre os casos de uso.

5. **Classificar chamado:** é a atividade de classificação dos chamados conforme apresentada na tabela 5.1.
6. **Justificar solicitação:** é a atividade que o usuário deve realizar, preenchendo o motivo da solicitação de manutenção, quando a estimativa da solicitação ultrapasse 8 horas de trabalho, que é enviada para aprovação da solicitação de manutenção.
7. **Aprovar solicitação:** é a atividade de aprovar a solicitação de manutenção feita pelo usuário. O comitê de informática analisa o motivo da solicitação, as horas estimadas para atendimento, o custo da manutenção e os benefícios gerados pela alteração do sistema. Esta fase é executada se a estimativa da solicitação ultrapassar a 8 horas de trabalho. Se a estimativa for inferior a 8 horas, a aprovação é automática.
8. **Priorizar chamado:** é a atividade de organização dos chamados de acordo com a prioridade, conforme apresentado na tabela 5.2.
9. **Implementar:** é a fase de trabalho técnico para atender a solicitação feita pelo usuário. Através da especificação de manutenção gerada, torna-se possível a implementação de uma solução para um problema apresentado pelo usuário.
10. **Apontar horas trabalhadas:** é a fase de apontamento do trabalho realizado, possibilitando gerar um banco de dados histórico, contendo o tempo estimado e o tempo trabalhado em uma atividade, possibilitando melhorar a estimativa e o gerenciamento de tempo a cada projeto.
11. **Testar:** é a fase de validação do trabalho implementado. O resultado é um sistema alterado e testado pronto para ser implantado em ambiente de produção.
12. **Implantar:** é a atividade de implantar todas as alterações necessárias para atender a solicitação feita e não impactar em outros pontos do sistema, que já estão em pleno funcionamento.
13. **Homologar:** é o compromisso que o usuário assume, indicando que as alterações feitas estão de acordo com as solicitadas e funcionando. A homologação é feita por e-mail pelo usuário, que é anexado ao chamado no

Fireman, encerrando o processo. As atividades são reiniciadas, caso o usuário não esteja de acordo, através de um novo chamado indicando que é retorno de homologação relacionado ao original.

5.4 Papéis dos participantes

A proposta possui os mesmos papéis do processo atual, adicionando novas atividades aos papéis existentes e incluindo o analista de qualidade e o comitê de informática.

- **Analista de suporte**: permanece com as mesmas funções anteriores, sendo responsável por receber as solicitações dos usuários, registrar o chamado e enviar para o coordenador de desenvolvimento continuar no fluxo do processo de manutenção;
- **Coordenador de desenvolvimento de sistemas**: foi adicionado à função de engenharia reversa, a utilização de métrica de software, gerenciamento de tempo de projeto, melhoria da estimativa de tamanho de software com a manutenção da base histórica de projeto do previsto e do realizado, priorização de chamados de até 8 horas de trabalho da empresa, alocação de chamados priorizados por função, além das funções anteriormente realizadas;
- **Comitê de informática**: é um novo papel nesta proposta, responsável por aprovar e priorizar as solicitações de manutenção de sistemas da empresa em geral de chamados que excedem 8 horas de trabalho;
- **Analista programador**: foi adicionada uma função de chamados por prioridades estabelecido pelo comitê de informática, que o analista deve seguir. Foi incluída uma função de apropriação de horas por atividade, onde o analista deve relacionar a hora trabalhada com a atividade realizada. Foi retirada a atividade de implantação do sistema em produção, sendo que, as demais funções permanecem idênticas ao processo atual;

- **Analista de Qualidade**: é um novo papel nesta proposta, responsável por executar os planos de testes de acordo com os casos de uso referente ao chamado e instalar as alterações em produção referente à solicitação do usuário após teste;
- **Homologador**: permanece com as mesmas funções anteriores, sendo responsável por acompanhar junto aos usuários se o chamado foi atendido conforme a solicitação do usuário. Estes chamados são enviados pelos analistas programadores, coordenador de desenvolvimento e analistas de qualidade;
- **Usuário**: permanece com as mesmas funções anteriores, sendo responsável por solicitar a manutenção de sistemas e de verificar se as solicitações estão de acordo com as alterações feitas, além de homologar o sistema permitindo a área de helpdesk encerrar o chamado do usuário;
- **Administrador de banco de dados**: permanece com as mesmas funções anteriores, sendo responsável por qualquer manutenção no banco de dados que, venha a ser necessária, em função das necessidades identificadas pelo analista programador que, seja solicitada pelo coordenador de desenvolvimento de sistemas.

Alguns papéis mantiveram-se inalterados, pois possuem responsabilidades bem definidas, permitindo um bom fluxo dentro do processo. Em outros casos, foram incluídas algumas responsabilidades identificadas, para melhoria da proposta apresentada. Os papéis incluídos são: comitê de informática e analista de qualidade, que complementam os papéis para um bom funcionamento do processo proposto.

5.5 Artefatos aplicáveis

Entre os artefatos propostos obrigatórios e independentes de processo estão o modelo de banco de dados sempre atualizado, a descrição dos casos de uso, os diagramas de caso de uso, o diagrama de classes e a ferramenta de controle de versão.

O modelo de banco de dados é sempre atualizado em qualquer manutenção no banco de dados, mas periodicamente é feita engenharia reversa do banco de dados físico para o modelo de dados através de ferramenta. É um artefato obrigatório para o processo proposto.

O caso de uso é um artefato da UML que na sua ausência é gerado através de engenharia reversa e é um artefato obrigatório para o novo processo de manutenção de software. Foi escolhido o caso de uso por ser de fácil utilização, muito utilizado no mercado. Além disso, a empresa desenvolve os sistemas baseados em componentes.

O diagrama de caso de uso é um artefato da UML que na sua ausência deve ser gerado, limitando-se ao escopo da manutenção solicitada.

O diagrama de classes é outro artefato da UML que deve ser sempre atualizado por engenharia reversa através de ferramenta.

A ferramenta de controle de versão também é muito utilizada, pois nela são documentadas as alterações feitas e quem as fez, evitando possíveis falhas de desenvolvimento.

5.6 Ferramenta indicada

O Fireman é a ferramenta indicada para gerenciamento da fase de manutenção de software, mas adicionando algumas funcionalidades que o software não oferece. O software é de fácil integração, pois utiliza banco de dados e de fácil entendimento do modelo de dados, que possibilita a criação de novas funcionalidades que foram identificadas para esta proposta:

- Função de priorização de chamados por analista programador para o coordenador de desenvolvimento de sistemas;
- Função de aprovação e priorização de chamados para o comitê de informática;
- Função de apontamento de horas trabalhadas relacionado às atividades enviadas ao papel destinado;
- Função de entrada de informações para justificar a solicitação de manutenção de software pelo usuário solicitante;
- Função de estimativa de horas por atividade dentro do chamado de acordo com a métrica de software adotada para o coordenador de desenvolvimento de sistemas;
- Função para consulta dos chamados pendentes e realizados para os supervisores e gerentes das áreas de negócio.

5.7 Processo de reengenharia

A engenharia reversa é aplicada nesta proposta com o objetivo de obter artefatos utilizados no processo de manutenção:

- modelo de banco de dados através de ferramenta específica;
- descrição dos casos de uso que devem ser escritos ou atualizados, de acordo com a manutenção, através da análise do código fonte e da

aplicação. O caso de uso faz parte da especificação de sistema utilizada para desenvolver a manutenção solicitada;

- diagramas de casos de uso que devem ser atualizados ou desenhados em relação a manutenção solicitada;
- diagrama de classes que deve ser atualizado a cada manutenção a partir do código através de ferramenta específica.

6. CONCLUSÃO

No mercado competitivo atual, uma preocupação constante das organizações é o cumprimento de cronogramas e o custo efetivo de seus projetos de manutenção de software.

Todas estas características dependem de um processo adequado de manutenção de software. Nesse processo é de fundamental importância a existência de artefatos básicos necessários à manutenção de software, assim como uma especificação de projeto definida e estimativas mais precisas de tamanho de software.

Diante desta necessidade, este trabalho teve como objetivo fornecer um processo de manutenção de software adequado a pequenos projetos, prevendo os benefícios advindos de se possuir uma documentação mínima, porém fundamental para atender as necessidades do sistema. Para atender a este objetivo foram definidos alguns pontos relevantes:

1. estudar o modelo em cascata e engenharia reversa;
2. estudar as principais métricas de estimativa de tamanho de software: análise por ponto de função e pontos por caso de uso;
3. analisar um processo de software vigente em uma empresa, onde pode ser avaliado e melhorado;
4. elaborar uma proposta de alteração do processo de manutenção vigente na empresa analisada.

Para atender ao segundo ponto foram estudadas as principais características da análise de pontos por função e pontos por caso de uso.

Para atender o terceiro ponto foi selecionada uma empresa que tinha um processo de manutenção de software de pequenos projetos em uso, que permitisse analisar e criticar o processo.

E quanto ao último ponto, foi elaborada uma proposta de alteração de processo de manutenção de software, onde é, proposto melhorias no processo estudado da empresa selecionada.

Portanto, como contribuições deste trabalho, podem ser destacadas:

- a proposição de um processo de manutenção de software adequado a pequenos projetos com os benefícios da definição de um conjunto de documentos;
- a proposição de métrica de software dentro de um processo de manutenção de software;
- a proposição de engenharia reversa no processo de manutenção de software.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, M. Pontos de função ou pontos de caso de uso? Como estimar projetos orientados a objetos. **Developers´magazine.** V. 7, n. 77. Jan., 2003.

BASILI. V.; CALDIERA, G.; ROMBACH, H. Goal Question Metric paradigm. In: **Encyclopedia of Software Engineering.** V. 2, 1994.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The unified modeling language user guide.** Reading: ADDISON-WESLEY. 1999.

DAMODARAN, M; WASHINGTON, A. Estimation using use case points. **Computer Science Program**. Texas–Victoria: University of Houston. S.d. Disponível em: http://bfpug.com.br acesso em: 30 de nov. 2004.

DEKKERS, C. Measuring the 'logical' or 'functional' size of software projects and software application. **ISO BULLETIN.** May, 2003.

FENTON, N., NEIL, M. Software Metrics: Roadmap. **Future of Software Engineering Limerick Ireland ACM.**, 2000.

FENTON; PFLEEGER, S. **Software metrics:** a rigorous & practical approach. Boston: PWS Publishing Company, 1997.

FIREMAN ENTERPRISE, versão 3.0: Software de gerenciamento de chamados: Sial Software, 2003.

FUREY, S. **Why we should use Function Points**. IEEE. Mar/April, 1997.

GARMUS, D., HERRON, D. **Function Point Analysis: Measurement Practices for successful for software projects**. Addison Wesley, 2000.

HAZAN, C. **Análise por pontos de função: uma abordagem gerencial**. Rio de Janeiro, SERPRO. 1999. 1 v.

IFPUG. International Function Point Users Group. **Function Point Counting Practices Manual**: Release 4.1, 2000.

FIPS PUBS; **Integration Definition For Function Modeling (IDEF0)**. Federal Information Processing Standarts Publications, 1993.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **Unified Software Development Process.** Addison-Wesley, 1996.

KARNER, G. **Use Case Points**: resource estimation for Objectory projects, 1993.

LONGSTREET, D. **Fundamentals of Function Point Analysis**. Longstreet Consulting Inc., 2002. Disponível em: http://www.ifpug.com/Articles/default.htm. Acesso em: 27 de nov. 2004.

MCPHEE, C. **Software process management**: software size estimation. University of Calgary. 1999. Disponível em: http://sern.ucalgary.ca. Acesso em: 30 de nov. 2004.

MISIC, V.; TESIC, D. Downsizing the estimation of software quality: a small object-oriented case study. In: Technology of Object-Oriented Languages and Systems. **Proceedings.** Disponível em: http://dblib2.computer.org. Acesso em: 10 nov. 2004.

PRESSMAN R. S. **Software Engineering A Practitioner's Approach**.: McGraw-Hill, 2001.

RIBU, K **Estimating object-oriented software projects with use cases.** Oslo, 2001 Tese (Mestrado-Departament of Informatics).

ROSS, M. Size does matter: continuous size estimating and tracking. **Quantitative software management,** 1999. Disponível http://www.qsm.com. Acesso em: 30 de nov. 2004.

SIAL SOFTWARE, **FIREMAN**. Disponível em http://www.fireman.com.br. Acesso em: 12 de dez. 2004.