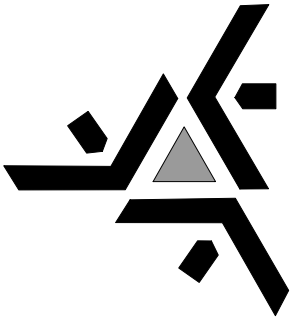

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática
Curso de Engenharia de Produção

Uso de Ferramentas e Metodologias para Garantir a Qualidade no Desenvolvimento de Software

Kleber Henrique Dias

TCC-EP-53-2006

Maringá - Paraná
Brasil

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática
Curso de Engenharia de Produção

Uso de Ferramentas e Metodologias para Garantir a Qualidade no Desenvolvimento de Software

Kleber Henrique Dias

TCC-EP-53-2006

Trabalho de conclusão de curso apresentado ao curso de Engenharia de Produção, do Centro de Tecnologia, da Universidade Estadual de Maringá.
Orientadora: *Prof.ª MSc. Maria de Lourdes Santiago Luz*

Maringá - Paraná
2006

Kleber Henrique Dias

Uso de Ferramentas e Metodologias para Garantir a Qualidade no Desenvolvimento de Software

Este exemplar corresponde à redação final do Trabalho de Conclusão de Curso aprovado como requisito parcial para obtenção do grau de Bacharel em Engenharia de Produção da Universidade Estadual de Maringá, pela comissão formada pelos professores:

_____

Orientadora: Profª.MSc. Maria de Lourdes Santiago Luz
Departamento de Informática, CTC

_____

Prof. MSc. Carlos Antonio Pizo
Departamento de Informática, CTC

Maringá, novembro de 2006

DEDICATÓRIA

Para a minha mãe, Genilda Ribeiro Dias, que me ensinou o que havia de mais importante para se aprender.

AGRADECIMENTOS

Agradeço à minha família, que me ensinou a dar os primeiros passos no longo caminho que me trouxe até aqui. Agradeço à minha namorada, Juliana, que me deu todo o carinho e suporte que precisei e soube lidar com o meu temperamento difícil dos últimos dias. Agradeço aos meus amigos, que fizeram esse caminho ser mais feliz. Agradeço a todos os professores que passaram pela minha vida, em especial aos que contribuíram com este trabalho. Agradeço à minha orientadora, a professora Maria de Lourdes, que aceitou o desafio de me orientar neste tema tão complicado. Finalmente agradeço a Deus, por permitir que tudo isso acontecesse.

RESUMO

O desenvolvimento de software carece do uso de ferramentas e metodologias durante o seu processo. Tais recursos existem e estão à mão dos desenvolvedores, que muitas vezes ignoram sua existência ou importância.Este trabalho visa mostrar como o uso de algumas ferramentas e metodologias, juntamente com o planejamento do projeto, contribuem para um software de melhor qualidade.O trabalho inicia-se com uma discussão sobre qualidade, seguindo com o uso de métricas e realização de estimativas. Prossegue com o planejamento do projeto, atividades para garantir a qualidade de software, os testes a serem realizados e a manutenção do mesmo após ser entregue. Por fim, o trabalho apresenta um estudo de caso realizado em uma empresa que pretende desenvolver uma nova versão de seu principal produto, aumentando a qualidade de seu software com o uso de algumas ferramentas aqui mostradas.

Palavras-Chave: Qualidade de software. Projeto de software. Engenharia de Software.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ISO: *International Organization for Standardization;* Organização Internacional para Padronização.

IEC: *International Electrotechnical Commission;* Comissão Internacional de Eletrônical.

CMM: *Capability Maturity Model;* Modelo de Maturidade de Capacidade.

CMMI: *Capability Maturity Model Integration;* Modelo de Maturidade de Capacidade Integrada.

SEI: *Software Engineering Institute;* Instituto de Engenharia de Software.

SW-CMM: *Capability Maturity Model for Software;* Modelo de Maturidade de Capacidade para Software.

SECM: *Systems Engineering Capability Maturity Model;* Modelo de Maturidade de Capacidade de Sistemas de Engenharia.

IPD-CMM: *Integrated Product Development Capability Maturity Model;* Modelo de Maturidade de Capacidade de Desenvolvimento Integrado de Produto.

MPS: Melhoria do Processo de Software.

LOC: *Lines of Code;* Linhas de Código.

KLOC: Mil Linhas de Código.

FP: *function point;* Pontos de Função.

CASE: *Computer-Aided Software Engineering;* Engenharia de Software com Suporte de Computador.

SQA: *Software Quality Assurance;* Garantia de Qualidade de Software.

SGCP: Sistema de Gestão Canônico-Pastoral.

1 INTRODUÇÃO

O processo de desenvolvimento de software, assim como o de qualquer outro produto, deve passar por etapas de planejamento com o uso de diversas ferramentas que facilitam sua posterior implementação e garantem um resultado final satisfatório.

Entretanto, a produção de software muitas vezes não passa por um planejamento prévio, com uma análise de como será o produto antes de se iniciar a sua produção. Alguns desenvolvedores consideram perda de tempo a realização de um planejamento prévio, acarretando numa enorme quantidade de softwares de baixíssima qualidade, produtos que não atendem aos clientes fazendo-os buscar outro software ou até mesmo desistir de um processo de informatização.

As metodologias e ferramentas de planejamento e desenvolvimento de produtos ditos convencionais são muitas vezes as mesmas utilizadas na obtenção de um bom software, com suas adaptações. Para Koscianski e Soares (2006, p. 18) “a qualidade de software ainda depende principalmente do correto emprego de boas metodologias pelos desenvolvedores.”

Ter uma visão do software como um produto que deve ter um planejamento e primar pela qualidade reduziria o número de softwares com baixa qualidade encontrados hoje no mercado, feitos principalmente por programadores autônomos.

Juran (1991) atribui a responsabilidade pela qualidade final do produto ou serviço, neste caso um software, à função qualidade, que segundo Juran (1991) "é o conjunto das atividades através das quais atingimos a adequação ao uso, não importando em que parte da organização estas atividades são executadas.”

No contexto de desenvolvimento de software, qualidade pode ser entendida como um conjunto de características a serem satisfeitas em um determinado grau, de modo que o produto de software atenda às necessidades explícitas e implícitas de seus usuários. Entretanto, não se obtém qualidade do produto de forma espontânea. Ela tem de ser construída. Assim, a qualidade do produto depende fortemente da qualidade de seu processo de desenvolvimento.

Este trabalho inicia-se com a abordagem da qualidade por vários autores. Prossegue com a gerência de projetos e métricas de software, abordando a definição do escopo do software e realização de estimativas. Continua com o planejamento de projetos, a análise de riscos e determinação de um cronograma. Após isso o trabalho aborda como garantir a qualidade do software, a realização dos testes e o que fazer após a entrega do produto ao cliente. Para finalizar, um estudo de caso em uma pequena empresa de desenvolvimento de software.

1.1 Objetivos

Este trabalho tem como objetivo o estudo de ferramentas e metodologias de qualidade que podem ser utilizadas no processo de desenvolvimento e manutenção de software, incluindo as etapas compreendidas, buscando um produto de qualidade que atenda às expectativas dos clientes.

1.2 Metodologia

O trabalho consiste de um estudo das ferramentas já disponíveis na engenharia de software para a obtenção de um software com qualidade. Para o desenvolvimento deste projeto foi realizado: Pesquisa bibliográfica e estudo de caso.

A pesquisa bibliográfica foi desenvolvida como fonte direta e indireta, para constituir um corpo teórico sobre o assunto que será abordado neste projeto. Foi desenvolvida com base em material já elaborado, constituído principalmente de livros e artigos científicos.

O estudo de caso é uma modalidade de pesquisa que consiste no estudo profundo e exaustivo de um ou poucos objetivos, de maneira que permita seu amplo e minucioso conhecimento, tarefa quase impossível mediante outros esboços já considerados. Foi estudado o processo de desenvolvimento de uma nova versão do sistema SGCP-Paróquia, um software específico para uso na Igreja Católica, desenvolvido pela empresa Theòs Informática. As especificações do software são baseadas em informações coletadas junto aos seus usuários por meio de questionários e da própria experiência dos desenvolvedores.

2 QUALIDADE

Para Weinberg (1993, p. 5), qualidade é a conformidade às exigências de alguma pessoa, sendo, portanto relativa. Esta é uma definição bem próxima da de Juran, que diz que qualidade é adequação ao uso (Juran, 1991) e também da de Crosby que diz que "a qualidade é a conformidade às exigências" (*apud* WEINBERG, 1993, p. 5).

Paladini (1994, p. 16) coloca que:

> *qualidade corretamente definida, é aquela que prioriza o consumidor. Isto mostra que a qualidade é mais do que simples estratégias ou técnicas estatísticas, é antes uma questão de decisão, que reflete em políticas de funcionamento da organização.*

2.1 Qualidade Total

Ao entrar na virada da primeira metade deste século, os japoneses sentiram que usando somente os métodos clássicos e estruturalistas de gestão seria muito difícil conseguir alcançar o desenvolvimento necessário para tornarem suas empresas criativas e competitivas. Diante deste cenário desfavorável, era imperativo encontrar um novo modelo de gestão que pudesse operar com agilidade e competência. Foi na busca deste novo modelo de organização que os japoneses encontraram nos ensinamentos do Dr. Deming a fórmula correta que, bem utilizada, provocaria as alterações profundas almejadas, tornando as empresas mais flexíveis, competitivas e conseqüentemente mais capazes de conquistar os mercados mundiais. Nasce desta forma um novo modelo de gestão que ficou conhecido mundialmente como qualidade total.

Adiante será descrita a qualidade total na visão de alguns autores. Cada um deles possui o seu ponto de vista, mas em todos é possível identificar as similaridades bem como a possibilidade de aplicação em diversos tipos de empresas, entre elas as de desenvolvimento de software. Conceitos como a grande participação de todos da empresa, envolvimento da alta gerência, motivação, organização, treinamento, manutenção, foco no cliente, entre outros, são muito válidos também na busca do software de qualidade, como será visto ao longo deste trabalho.

2.1.1 Abordagem na visão de Deming

A abordagem de Deming é baseada no uso de técnicas estatísticas para reduzir custos e aumentar a produtividade e a qualidade. Para descrever sua filosofia, Deming (1990) definiu 14 pontos:

a) criar uma constância de propósitos de melhorar produtos e serviços;

b) adotar a nova filosofia é o momento de iniciar um movimento por mudanças;

c) deixar de contar com a inspeção em massa, a qualidade não se origina da inspeção, mas do melhoramento do processo;

d) acabar com o sistema de compras baseado apenas no preço;

e) melhorar constantemente o sistema de produção e serviço;

f) lmplantar métodos modernos de treinamento no trabalho;

g) lmplantar métodos modernos de supervisão, instituir a liderança;

h) expulsar o medo da organização;

i) romper as barreiras entre as áreas e staff;

j) eliminar slogans, exortações e metas para a mão-de-obra, sem, no entanto, oferecer meios para alcançá-las;

k) eliminar os padrões de trabalho e cotas numéricas;

l) eliminar as barreiras que privam o empregado de ter orgulho do seu trabalho;

m) retreinamento contínuo;

n) criar uma estrutura na alta administração que tenha como função implantar os 13 pontos anteriores.

O enfoque de Deming está no controle e melhoria de processo, não apresentando para tanto um sistema estruturado ou uma metodologia clara para a implementação de sua abordagem nas empresas.

2.1.2 Abordagem na visão de Juran

As principais contribuições de Juran (1991) foram na definição e organização dos custos da qualidade e no enfoque da qualidade como uma atividade administrativa. Juran (1991) atribui a responsabilidade pela qualidade final do produto ou serviço à função qualidade, que segundo Juran (1991) compreende nas atividades que possibilitam atingir a adequação ao uso.

Para garantir que a função qualidade seja executada de modo a atingir os melhores resultados, Juran (1991) propôs uma trilogia de atividades:

a) Planejamento da Qualidade;

b) Controle da Qualidade;

c) Aperfeiçoamento da Qualidade.

A implantação desta abordagem é baseada na formação de equipes de projeto para a resolução de problemas, um de cada vez, propiciando um melhoramento contínuo da qualidade. Não há, no entanto, uma preocupação em organizar todas as atividades da função qualidade, de modo a garantir os melhores resultados desde o início do processo.

2.1.3 Abordagem na visão de Crosby

Crosby (1985), o pai da filosofia “Zero Defeito”, se baseia na teoria de que a qualidade é assegurada se todos se esforçarem em fazer seu trabalho corretamente da primeira vez. Para Crosby (1985), a qualidade é responsabilidade dos trabalhadores. O autor não considera, no entanto, outros aspectos que afetam a qualidade e que estão fora do controle dos operários, como os problemas com a matéria-prima, erros de projeto entre outros.

Crosby (1985) instituiu seus 14 pontos, que constituem as etapas de implementação de sua abordagem. São eles:

a) dedicação da alta gerência e comprometimento por meio da elaboração de um documento com a política e os objetivos da empresa;

b) constituição de equipes para melhorias coordenadas pelos gerentes;

c) medição dos resultados;

d) avaliação dos custos da qualidade;

e) comunicação dos resultados aos supervisores e operários;

f) reunião para identificação dos problemas;

g) estabelecimento de um comitê informal para a divulgação do programa;

h) treinamento da gerência e supervisão;

i) instauração do dia zero defeitos, onde os resultados anuais são divulgados e efetua-se o reconhecimento a todos os participantes do programa;

j) estabelecimento dos objetivos a serem seguidos;

k) consulta aos operários sobre a origem dos problemas;

l) recompensa àqueles que atingiram os seus objetivos;

m) formação dos conselhos da qualidade;

n) etapa final: faça tudo de novo.

Sua filosofia é voltada mais para o comportamento humano como único meio para se garantir a qualidade. O comportamento humano "Zero Defeito" é conseguido por meio de motivação e exortações. Esta abordagem em curto prazo pode atingir alguns resultados positivos, no entanto, em longo prazo a motivação das pessoas acaba diminuindo e a sustentação do programa de qualidade fica comprometida. É necessário que haja meios bem definidos, com

uma metodologia bem estruturada, para garantir o sucesso do programa e a conquista da qualidade total.

Weinberg (1993) afirma que as idéias de Philip Crosby podem ser aplicadas ao software, mas com algumas modificações. Segundo Weinberg (1993), Crosby define a qualidade como a conformidade com os requisitos. Para Weinberg (1993, p. 39), "em software, a conformidade com os requisitos não é suficiente para definir a qualidade de software, porque os requisitos não podem ser tão exatos como numa operação de manufatura."

2.1.4 Abordagem na visão de Feigenbaun

Feigenbaun ficou conhecido pela introdução do termo *Total Quality Control* (TQC) em 1961. Em sua abordagem, a qualidade deixa de ser responsabilidade de um departamento especializado em controle da qualidade e passa a ser função de todas as áreas da empresa. Para coordenar as atividades de todas as áreas da empresa no controle da qualidade, Feigenbaun (1961) sugere uma estrutura sistêmica.

Segundo Feigenbaun (1961), seria necessário a integração dos esforços relativos ao desenvolvimento, manutenção e melhoria da qualidade com todos os setores da organização, como o marketing, engenharia, produção e serviços, de forma que todos desempenhem suas atividades da forma mais econômica possível, visando primeiramente atender plenamente às necessidades do consumidor.

2.1.5 Abordagem na visão de Ishikawa

A abordagem de Ishikawa (1993) nasceu a partir da compilação de diversos aspectos do trabalho de vários especialistas como Deming, Juran e Shewart, acrescendo a eles uma grande preocupação com a participação do elemento humano e trazendo para o controle da qualidade uma visão humanística sob a influência dos trabalhos de Maslow, Herzberg e McGregor.

Sua filosofia é voltada para a obtenção da qualidade total (qualidade, custo, entrega, moral e segurança) com a participação de todas as pessoas da organização, da alta gerência aos

operários do chão de fábrica. No TQC japonês, por meio de uma metodologia bem definida, todos os níveis empresariais colocam suas atividades diárias sob controle, garantindo a qualidade por toda a empresa.

Ishikawa (1993) enfatiza também a participação dos funcionários por meio dos círculos de controle de qualidade (CCQ), objetivando a melhoria contínua dos níveis de qualidade e resolução de problemas. A abordagem de Ishikawa (1993) exige por parte da empresa um comprometimento e uma mobilização significativamente maiores do que nos outros modelos.

Em algumas vezes, chega-se a relacionar esta necessidade de persistência e entusiasmo com a cultura e tradição japonesas, o que também acaba gerando dúvidas quanto à validade desta abordagem em países ocidentais.

2.2 Qualidade de Software

Para Koscianski e Soares (2006, p. 18) "a qualidade de software ainda depende principalmente do correto emprego de boas metodologias pelos desenvolvedores. Embora eles sejam apoiados por várias ferramentas, ainda restam problemas sérios sem tal suporte."

Para Pfleeger (2004, p. 8), além do usuário, ou consumidor, o software deve também ser julgado pelos que realizam o projeto, pelos que escrevem o código e pelos que fazem a manutenção dos programas já prontos.

Segundo Weinberg (1993, p. 11) a definição comum de qualidade sendo a ausência de erros e que dominou o pensamento de qualidade de software por muitos anos é totalmente inadequada. Isso leva o pessoal de desenvolvimento a ignorar requisitos para melhorar a qualidade. Além disso, dependendo da complexidade do software, é impossível eliminar todos os erros, visto que há um determinado limite em que o custo para se eliminar um erro é maior que o custo de conviver com o mesmo.

Koscianski e Soares (2006, p. 22) enumeram uma lista dos principais problemas enfrentados na construção e utilização de software:

- cronogramas não observados;
- projetos com tantas dificuldades que são abandonados;
- módulos que não operam corretamente quando combinados;
- programas que não fazem exatamente o que era esperado;
- programas tão difíceis de usar que são descartados;
- programas que simplesmente param de funcionar.

Os itens dessa lista não são problemas novos, mas sim problemas que já se faziam presentes no fim da década de 60. A produção de software com qualidade, utilizando ferramentas de auxílio e técnicas de planejamento de projetos, precisa superar os problemas dessa lista para ser bem sucedida. (KOSCIANSKI e SOARES, 2006)

2.3 ISO 9000 Aplicada ao Software

Segundo Weber, Rocha e Nascimento (2001, p. 10), a ISO reconhece quatro diferentes categorias de produtos:

a) Produtos (hardware): ISO 9004-1;

b) Serviços: ISO 9004-2;

c) Materiais processados: ISO 9004-3;

d) Software: ISO 9000-3.

É difícil implantar os requisitos da ISO 9001 ou da ISO 9002 em software, pois a terminologia é muito voltada para hardware. Por isso recomenda-se o uso da ISO 9000-3. Outra norma importante é a ISO/IEC 12207 que trata dos processos que compõem o ciclo de vida do software e a ISO/IEC TR 15504 que trata de tecnologia da informação.

Para a empresa, conquistar a certificação ISO 9000 significa alcançar padrão internacional de qualidade em seus processos de software.

2.4 CMM Aplicado à Melhoria dos Processos de Software

O CMM – *Capability Maturity Model* foi desenvolvido pelo SEI – *Software Engineering Institute*. Ele foi baseado nos conceitos de qualidade total de Crosby, onde ele demonstra que a implantação de sistemas da qualidade em empresas segue um amadurecimento gradativo, a saber: incerteza, despertar, esclarecimento, sabedoria e certeza. (Weber, Rocha e Nascimento, 2001, p. 15)

O modelo CMM tem como objetivo estabelecer – com base em estudos históricos e conhecimento operacional – um conjunto de "melhores práticas" que devem ser utilizadas para um fim específico. O CMM foi dividido em cinco níveis referentes à maturidade que a organização possui. São eles: inicial, repetitivo, definido, gerenciado e otimizado. Cada nível é dividido em áreas-chave de processo. Cada área-chave é detalhada nas práticas-chave, que são os quesitos a serem cumpridos na implantação do modelo. (Weber, Rocha e Nascimento, 2001, p. 16)

As organizações que se encontram no nível 1 do CMM podem desenvolver software de alta qualidade, mas seu desempenho depende da competência das pessoas. A ISO 9000 equivaleria ao nível 3 do CMM.

A implantação do CMM é um processo de longo prazo, pois envolve aspectos de mudança cultural. A alta administração tem um papel fundamental, pois deve demonstrar uma postura clara de interesse no andamento e cobrar quando necessário. (Weber, Rocha e Nascimento, 2001, p. 18)

Com o passar do tempo surgiram vários modelos CMM. Para suprir as limitações do CMM, unificar os vários modelos CMM existentes e implementar melhorias a partir das experiências adquiridas até ali, surgiu o CMMI (*Capability Maturity Model Integration*), uma evolução do CMM que procura estabelecer um único modelo para o processo de melhoria corporativo, integrando diferentes modelos e disciplinas.

O CMMI foi criado a partir da união de 3 modelos, como é ilustrado na Figura 1:

a) *The Capability Maturity Model for Software* (SW-CMM) v2.0 draft C;

b) *The Systems Engineering Capability Maturity Model* (SECM);

c) *The Integrated Product Development Capability Maturity Model* (IPD-CMM) v0.98.

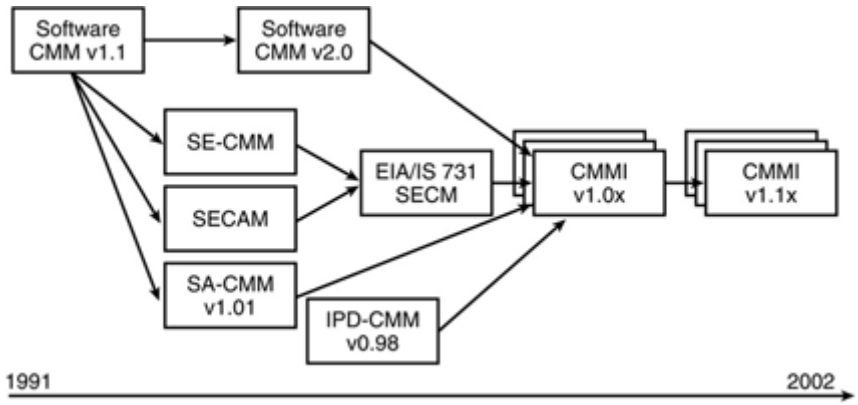

Figura 1: Modelos que diretamente influenciaram o CMMI.
Fonte: KULPA, 2003

Existem dois tipos de modelos do CMMI: o contínuo e o em estágios, que diferem na maneira de implementar, mas basicamente têm o mesmo conteúdo.

2.5 O Projeto MPS.BR

De acordo com Weber et al (2005), o projeto MPS.BR, Melhoria do Processo de Software Brasileiro, visa à melhoria de processo de software brasileiro e a um custo acessível. Compreende duas metas:

a) desenvolvimento e aprimoramento do modelo MPS, compatível com o CMMI e em conformidade com as normas ISO/IEC 12207 e ISO/IEC 15504;

b) a implementação e avaliação do modelo MPS a um custo acessível principalmente em pequenas e médias empresas.

O projeto MPS não tem o objetivo de definir algo novo. Sua novidade está em ser voltado para a realidade brasileira. Ele compreende o desenvolvimento de um Modelo de Referência

de Processo, de um Modelo de Avaliação de Processo e de um Modelo de Negócio para Melhoria de Processo de Software, MR-MPS, MA-MPS e MN-MPS, respectivamente. (WEBER et al, 2005)

3 GERÊNCIA DE PROJETOS E MÉTRICAS DE SOFTWARE

Nos processos de engenharia é muito comum medir as coisas para saber algo sobre elas. Assim também é o software, que deve ser medido para que se possa saber algo sobre ele, como o seu custo estimado, tempo de desenvolvimento, o esforço para o desenvolvimento, os riscos, e também para podermos comparar as medições com outros softwares.

Segundo Pressman (2005, p. 83) "a medição possibilita que gerentes e profissionais entendam melhor o processo de engenharia de software e o produto (software) que ele produz."

3.1 O Processo de Gerência de Projetos

A gerência de projetos é a primeira camada do processo de engenharia de software. Pressman (2005, p. 55) diz:

> *Para conduzir um projeto de software bem-sucedido, devemos compreender o escopo do trabalho a ser feito, os riscos em que incorreremos, os recursos exigidos, as tarefas a serem executadas, os marcos de referência a serem acompanhados, o esforço (custo) despendido e a programação a ser seguida.*

A gerência de projetos começa antes do trabalho técnico e encerra somente quando o software é aposentado. Pressman (2005, p. 55) continua:

> *Antes que um projeto possa ser planejado, os objetivos e o escopo devem ser estabelecidos, soluções devem ser consideradas e as restrições administrativas e técnicas, identificadas. Os objetivos identificam as metas globais do projeto sem levar em consideração como essas metas serão atingidas. O escopo identifica as funções primárias que o software deve realizar e, o que é mais importante, tenta delimitar essas funções de uma forma quantitativa.*

Medições e métricas ajudam a entender o processo técnico usado para se desenvolver um produto, bem como também o próprio produto. O processo é medido para melhorá-lo, enquanto o produto é medido para aumentar sua qualidade.

Quando um software é planejado, estimativas de esforço humano, duração e custo são feitas, muitas vezes usando-se a experiência de projetos passados. Ao se fazer estimativas deve-se estabelecer o escopo antecipadamente. Com o uso de métricas de software e o histórico de

medições passadas são feitas estimativas. O projeto é dividido em pequenas partes que serão estimadas individualmente. (PRESSMAN, 2005, p. 57)

A análise de riscos é crucial para o bom gerenciamento de projeto de software. Em seu livro sobre gerenciamento da engenharia de software, Tom Gilb (*apud* PRESSMAN, 2005, p. 58) diz: "Se você não atacar ativamente os riscos [técnicos e de projeto], eles lhe atacarão ativamente."

Quanto à programação do projeto de software, Pressman (2005, p. 58) diz:

> *A programação do projeto de software não é, de fato, absolutamente diferente de qualquer projeto de engenharia. Um conjunto de tarefas de projeto é identificado. Interdependências entre as tarefas são estabelecidas. O esforço associado a cada tarefa é estimado. Pessoas e outros recursos são atribuídos. Uma "rede de tarefas" é criada. Um gráfico de Gant (*Time-line*) é desenvolvido.*

Durante o desenvolvimento, a monitoração e controle serão feitos pelo gerente de projetos, anotando e acompanhando cada tarefa da programação e realizando os ajustes necessários.

3.2 Métricas de Produtividade e Qualidade de Software

Dá-se importância principalmente a métricas de produtividade e de qualidade – medidas do resultado do desenvolvimento de software em função do esforço na adequação ao uso. Há o interesse histórico que possibilita comparações com projetos passados.

3.3 Medidas do Software

No mundo da engenharia de software há dificuldades para se decidir sobre o que se medir e como avaliar essas medidas. Pressman (2005, p. 60) diz sobre medidas do software:

> *O software é medido por muitas razões: (1) indicar a qualidade do produto; (2) avaliar a produtividade das pessoas que produzem o produto; (3) avaliar os benefícios (em termos de produtividade e qualidade) derivados de novos métodos e ferramentas de software; (4) formar uma linha básica para estimativas; (5) ajudar a justificar os pedidos de novas ferramentas ou treinamento adicional.*

As medidas de software podem ser diretas, como custo e esforço, ou indiretas, como qualidade e complexidade, sendo as diretas mais fáceis de serem obtidas.

As métricas de software podem ser divididas ainda em métricas da produtividade, que se concentram na saída do processo de engenharia de software, métricas da qualidade que medem a adequação ao uso, e métricas técnicas que se concentram na característica do software e não no processo de desenvolvimento. Pode-se dividir uma segunda vez ainda em métricas orientadas ao tamanho, usadas para compilar as medições diretas da saída e da qualidade da engenharia de software, métricas orientadas para a função oferecem medidas indiretas e métricas orientadas às pessoas que compilam informações sobre a maneira segundo a qual as pessoas desenvolvem software de computador e percepções humanas sobre a efetividade das ferramentas e métodos. (PRESSMAN, 2005, p. 61)

A Figura 2 ilustra essa divisão das métricas de software proposta por Pressman (2005).

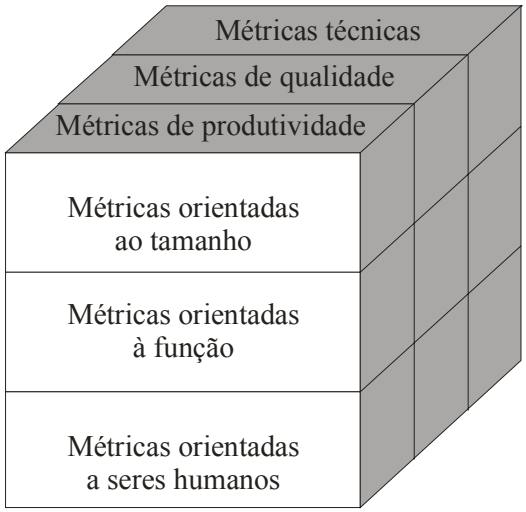

Figura 2: Divisão das métricas em categorias.
Fonte: PRESSMAN, 2005, p. 62

3.3.1 Métricas orientadas ao tamanho

São medidas diretas do software e do seu processo de desenvolvimento. É simples montar uma tabela de dados orientados ao tamanho, podendo serem usadas medidas como o KLOC (mil LOC, linhas de código, *Lines of Code*), número de páginas de documentação, número de

erros encontrados após entrega ao cliente, número de pessoas envolvidas, custos. A partir destes dados, outras métricas de qualidade e produtividade podem ser obtidas. Por exemplo:

$$\text{Produtividade} = \text{KLOC} / \text{pessoas-mês} \quad (1)$$

$$\text{Qualidade} = \text{Defeitos} / \text{KLOC} \quad (2)$$

$$\text{Custo} = \$ / \text{LOC} \quad (3)$$

Essas métricas não são universalmente aceitas. Seus defensores afirmam que as LOC estão em todos os projetos, facilmente contadas e com vasta literatura a respeito. Por outro lado, as medidas LOC dependem da linguagem de programação utilizada, penalizam programas bem estruturados e mais curtos e sua estimativa exige um nível de detalhes que pode ser difícil de conseguir. (PRESSMAN, 2005, p. 63)

3.3.2 Métricas orientadas à função

São medidas indiretas do software e do processo por meio do qual ele é desenvolvido que, ao invés de analisar linhas de código, concentram-se na "funcionalidade" ou "utilidade" do programa. Proposta pela primeira vez por A. J. Albrecht, sugerindo uma abordagem chamada método do ponto-por-função (*function point*). Os pontos de função (FPs) são derivados por uma relação empírica baseada em medidas de informações e complexidade do software.

Os pontos-por-função são computados completando-se uma tabela com cinco características do domínio de informação. Seus valores são definidos por Pressman (2005, p. 64) da seguinte forma:

a) número de entradas do usuário: Cada entrada do usuário que proporcione dados distintos orientados à aplicação é contada;

b) número de saídas do usuário: Saída refere-se a relatórios, telas, mensagens, etc. Cada saída do usuário que proporcione informações orientadas à aplicação é contada;

c) número de consultas do usuário: Uma consulta é definida como uma entrada *on-line* que resulta numa saída *on-line*;

d) número de arquivos: Cada agrupamento lógico de dados, podendo ser uma parte de um grande banco de dados ou um arquivo convencional;

e) número de interfaces externas: Todas as interfaces legíveis por máquina que sejam usadas para transmitir informações a um outro sistema.

Um valor de complexidade é definido para cada contagem, podendo ser simples, média ou complexa, sendo esta uma determinação um tanto subjetiva. Os pontos são determinados por Pressman (2005, p. 65) pela seguinte relação:

$$FP = \text{contagem total X } [0{,}65 + 0{,}01 \text{ X SOMA}(Fi)] \qquad (4)$$

onde a contagem total é a soma das FP, Fi são os valores de complexidade (i = 1 a 14). As constantes e os pesos são determinados empiricamente.

A Figura 3 exemplifica o uso da métrica ponto-por-função.

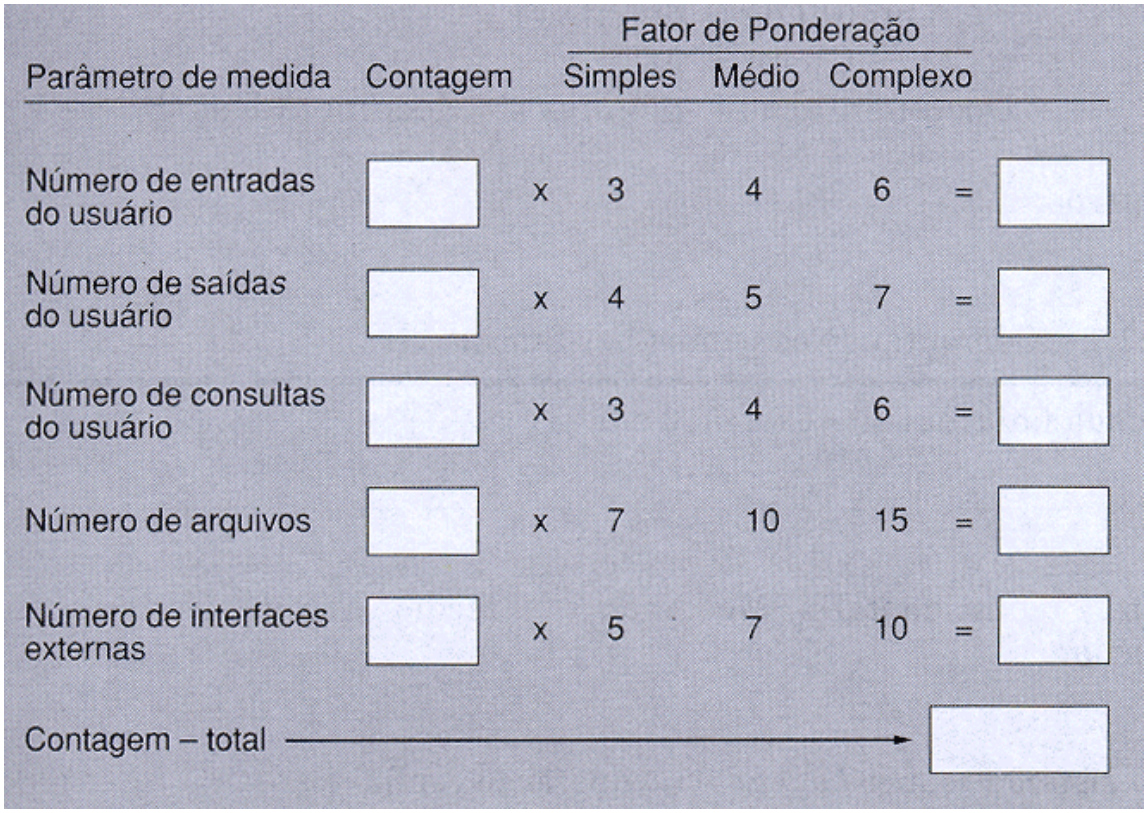

Figura 3: Computando a métrica ponto-por-função (*function point*).
Fonte: PRESSMAN, 2005, p. 65.

Os FP poderão ser usados analogamente às LOC como medida de produtividade, qualidade e outros. Assim, Pressman (2005, p. 66) define:

$$\text{Produtividade} = \text{FP} / \text{pessoa-mês} \quad (5)$$

$$\text{Qualidade} = \text{defeitos} / \text{FP} \quad (6)$$

$$\text{Custo} = \$ / \text{FP} \quad (7)$$

$$\text{Documentação} = \text{páginas de documentação} / \text{FP} \quad (8)$$

Para aplicações de engenharia de software e de sistemas, aplicações em que a complexidade algorítmica é elevada, utilizam-se os pontos de particularidades (*feature point*) onde se consideram os algoritmos, sendo algoritmo definido como um problema computacional delimitado que está incluído em um programa de computador específico. Os pontos de particularidade e os pontos de função representam a mesma coisa, a funcionalidade ou utilidade do software, porém para sistemas de tempo real mais complexos o valor da contagem dos pontos de particularidade geralmente se situa entre 20 e 35% maior do que por pontos-por-função (PRESSMAN, 2005, p. 67).

Os defensores do uso de FP afirmam que ela independe da linguagem e se baseia em dados que têm maior probabilidade de serem conhecidos no começo do projeto, facilitando uma estimativa. Seus opositores afirmam que o método se baseia parcialmente em dados subjetivos, que os dados sobre o domínio da informação podem ser difíceis de serem compilados mais tarde e que o FP não tem nenhum significado físico, sendo apenas um número.

3.4 Métricas de Qualidade do Software

A qualidade pode ser medida ao longo do processo de engenharia de software e depois da entrega do mesmo. “Métricas derivadas antes que o software seja entregue oferecem uma base quantitativa para se tomar decisões referentes a projeto e testes. Métricas usadas após a entrega concentram-se no número de defeitos e na manutenibilidade do sistema.” PRESSMAN (2005, p. 68).

McCall e Cavano (*apud* PRESSMAN, 2005, p. 69) definiram um conjunto de fatores da qualidade que avaliavam o software a partir de três pontos de vista: (1) operação do produto (usando-o); (2) revisão do produto (mudando-o); e (3) transição do produto (migrando-o).

3.4.1 Medidas da qualidade

As medidas *a posteriori* são as mais usadas, incluindo corretitude, manutenibilidade, integridade e usabilidade. Gilb (*apud* PRESSMAN, 2005, p. 70) sugeriu as seguintes definições para cada uma:

Corretitude. É o grau em que o software executa a função que dele é exigida. A medida mais comum são os defeitos por KLOC, sendo defeito a falta verificada de conformidade aos registros.

Manutenibilidade. É a facilidade com que um programa pode ser corrigido, adaptado ou ampliado. Não há formas de medições diretas. Indiretamente, uma métrica simples é o tempo médio para a mudança (MTTC), sendo este o tempo que demora para analisar o pedido de mudança, projetar uma modificação adequada, implementar a mudança, testá-la e distribuí-la.

Integridade. Mede a capacidade que um sistema tem de suportar ataques à sua integridade, feitos aos programas, dados e documentos. Para medir a integridade utiliza-se de dois atributos: ameaça e segurança. Ameaça é a probabilidade de um ataque ocorrer. Segurança é a probabilidade deste ataque ser repelido. Assim, a integridade pode ser definida como:

$$\text{Integridade} = \sum[1 - \text{ameaça} \times (1 - \text{segurança})] \qquad (9)$$

Usabilidade. Tentativa de se quantificar quão amigável é um software. Pode ser medida segundo quatro características: (1) habilidade física e/ou intelectual para se aprender o sistema; (2) o tempo para se tornar moderadamente eficiente no uso do sistema; (3) o aumento de produtividade quando o sistema é usado por alguém que seja moderadamente eficiente; (4) uma avaliação subjetiva das atitudes dos usuários em relação ao sistema.

3.5 Conciliando Diferentes Abordagens de Métricas

A relação entre linhas de código e pontos de função depende da linguagem de programação usada. Uma série de estudos tem tentado relacionar as medidas de FP e as de LOC. Pressman (2005) desaconselha que os dados de LOC / pessoa-mês (ou de FP / pessoa-mês) sejam usados para se fazer comparações entre um grupo e outro, ou para avaliar o desempenho das pessoas. Muitos fatores influenciam a produtividade, contribuindo para análises mal-interpretadas.

Basili e Zelkowitz (*apud* PRESSMAN, 2005, p. 73) definem cinco fatores importantes que influenciam a produtividade:

> *'Fatores humanos:' O tamanho e a experiência da organização de desenvolvimento.*
> *'Fatores do problema:' A complexidade do problema a ser resolvido e o número de mudanças nos requisitos ou restrições de projeto.*
> *'Fatores do processo:' Técnicas de análise e projeto que são usadas, linguagens e ferramentas CASE (Computer-Aided Software Engineering) disponíveis e técnicas de revisão.*
> *'Fatores do Produto:' Confiabilidade e desempenho do sistema baseado em computador.*
> *'Fatores relacionados a recursos:' Disponibilidade de ferramentas CASE, recursos de hardware e software.*

3.6 Argumentos para Métricas de Software

Segundo Pressman (2005, p. 76) é importante medir o processo de engenharia de software e o produto por que senão "não haverá nenhuma maneira real de determinarmos se estamos ou não melhorando." Com as medições podem ser estabelecidas metas de melhorias, estabelecendo uma linha básica de processo (*baseline*) a partir da qual as melhorias possam ser avaliadas.

3.6.1 Estabelecimento de uma linha básica (*Baseline*)

Quando se estabelece uma linha básica de métricas, benefícios podem ser obtidos a nível estratégico, técnico e de projeto. A linha básica consiste de dados compilados de projetos passados de desenvolvimento de software. Os dados da linha básica devem ter os seguintes atributos: (1) os dados devem ser razoavelmente precisos; (2) devem ser obtidos de tantos projetos quanto for possível; (3) as medições devem ser consistentes; (4) as aplicações devem ser idênticas ao trabalho que será estimado (PRESSMAN, 2005, p. 77).

A coleta de dados requer uma investigação histórica dos projetos passados para se reconstruir os dados exigidos. Assim que coletados, a computação das métricas é possível. Os dados computados devem ser avaliados e aplicados na estimativa. A avaliação dos dados concentra-se nas razões subjacentes para os resultados obtidos. As médias computadas são pertinentes ao projeto? Quais circunstâncias invalidam certos dados? (PRESSMAN, 2005, p. 78).

A Figura 4 ilustra o processo de coleta de métricas de software.

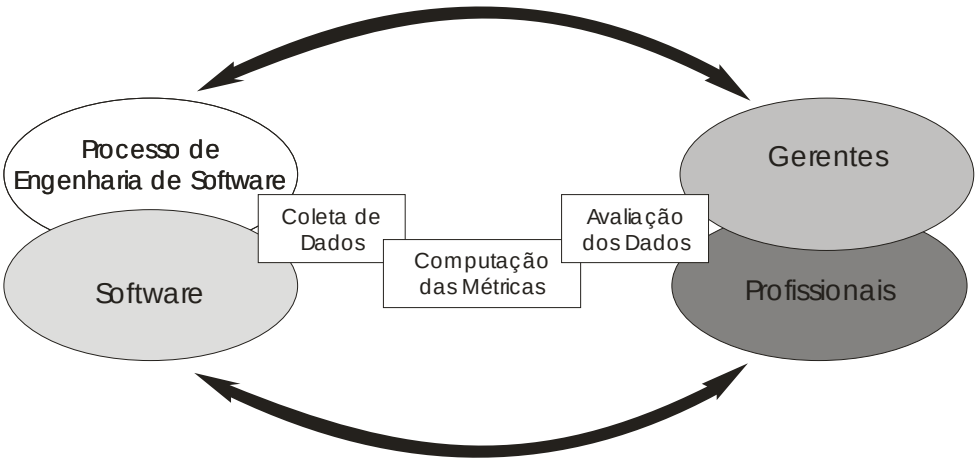

Figura 4: Processo de coleta de métricas de software.
Fonte: PRESSMAN, 2005, p. 78.

4 REALIZANDO ESTIMATIVAS

Segundo Pressman (2005, p. 125):

> *O planejador do projeto de software deve estimar três coisas antes que um projeto se inicie: quanto tempo ele durará, quanto esforço será exigido e quantas pessoas estarão envolvidas. Além disso, o planejador deve prever os recursos (hardware e software) necessários e os riscos envolvidos.*

A administração de projetos de software inicia-se com a realização de estimativas. Sobre isso, Pressman (2005, p. 89) diz:

> *A estimativa de recursos, custo e programação de atividades para um esforço de desenvolvimento de software exige experiência, acesso a boas informações históricas e a coragem para se comprometer com medidas quantitativas quando dados qualitativos forem tudo o que existir.*

A complexidade do projeto, o tamanho do projeto e o grau de estrutura do projeto afetam a precisão e eficácia das estimativas. Para Pressman (2005, p. 90), "a palavra estrutura refere-se à facilidade com que as funções podem ser dispostas em compartimentos e à natureza hierárquica das informações que devem ser processadas." À medida que o grau de estrutura aumenta, melhora a precisão e diminuem os riscos.

A disponibilidade de informações históricas e métricas de projetos passados ajuda a realização de estimativas com mais segurança. Segundo Pressman (2005, p. 91) "o objetivo do planejamento de projetos de software é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos." Para obter uma estimativa precisa o planejador deverá comparar pelo menos duas das técnicas descritas mais adiante.

4.1 Determinação do Escopo do Software

A determinação do escopo do software é a primeira atividade no planejamento de projetos de software que ajuda o planejador a desenvolver estimativas. Segundo Pressman (2005, p. 92) "o escopo do software descreve a função, o desempenho, as restrições, as interfaces e a confiabilidade." Às vezes pode ser interessante decompor as funções para facilitar a realização de estimativas.

A função, o desempenho e as restrições devem ser avaliados em conjunto. O planejador também deverá considerar as interfaces de hardware, software, pessoas e procedimentos com que o software interage.

Por ser difícil de se determinar a confiabilidade de um software o planejador pode se utilizar da natureza do projeto (por exemplo, um controle de estoques ou um sistema de tráfego aéreo) para ajudar a formular estimativas de esforço e de custo para se garantir a confiabilidade.

4.2 Estimativa dos Recursos

A estimativa dos recursos é a segunda tarefa de planejamento de software. Segundo Pressman (2005) os recursos são como a pirâmide da Figura 5, tendo na base as ferramentas e no topo as pessoas. “Cada recurso é especificado segundo quatro características: descrição do recurso, uma declaração da disponibilidade, tempo cronológico em que o recurso será exigido e por quanto tempo o recurso será aplicado.” (PRESSMAN, 2005, p. 96).

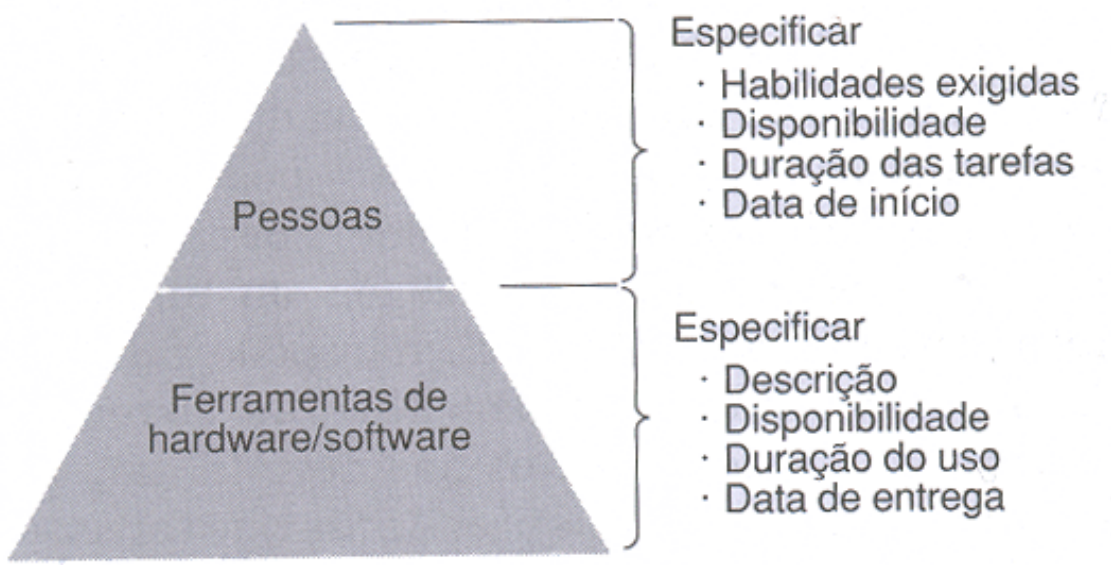

Figura 5: Recursos.
Fonte: PRESSMAN, 2005, p. 96.

4.2.1 Recursos humanos

O planejador avalia o escopo do software para definir as habilidades que serão necessárias no seu desenvolvimento. Para determinar o número de pessoas exigido deverá primeiro ser feito uma estimativa do esforço de desenvolvimento.

4.2.2 Recursos de hardware

Deve ser considerado o hardware onde se faz o desenvolvimento, o hardware de produção (destino – *target machine*) e outros elementos de hardware, como algum elemento de teste.

4.2.3 Recursos de software

Atualmente os engenheiros de hardware usam um conjunto de ferramentas de engenharia de software auxiliada por computador (CASE – Computer-Aided Software Engineering) análogas às ferramentas CAD/CAE que são utilizadas por engenheiros de hardware. Segundo Pressman (2005, p. 98) as ferramentas CASE podem ser divididas em algumas categorias principais:

Ferramentas de Planejamento de Sistemas de Informações: Proporcionam um meta-modelo a partir do qual sistemas de informações específicos são derivados. Ajudam aos desenvolvedores encaminhar os dados àqueles que precisam da informação. No fim, a transferência de dados é melhorada e as tomadas de decisão são agilizadas.

Ferramentas de Gerenciamento de Projetos: Ajuda o gerente a gerar estimativas de esforço, custo e duração de um projeto, definir uma divisão do trabalho, uma programação de atividades, monitorar os projetos e compilar métricas para estabelecimento de uma *baseline*.

Ferramentas de Apoio: Várias ferramentas de produção de documentos, rede, bancos de dados, comunicação e gerenciamento de configuração.

Ferramentas de Análise e Projeto: Ajudam na criação de um modelo e avaliação de sua qualidade e também a eliminar erros.

Ferramentas de Programação: Utilitários básicos, editores, compiladores e depuradores, ferramentas orientadas a objetos, bancos de dados e muitas outras ferramentas utilizadas em computadores pessoais.

Ferramentas de Integração e Testes: Ferramentas que oferecem uma variedade de diferentes níveis de apoio aos passos de teste de software. Podem ajudar a reduzir a quantidade de esforço do processo de teste.

Ferramentas de Construção de Protótipos e Simulação: Concentram-se na criação de telas e relatórios que ajudarão o usuário a entender o domínio de entrada e saída do sistema, permitindo uma análise do software antes mesmo que ele seja construído.

Ferramentas de Manutenção: Facilita processos de engenharia reversa e reengenharia, não dispensando, porém, o trabalho do engenheiro.

Ferramentas de *Framework*: Disponibilizam uma plataforma a partir da qual um ambiente integrado de suporte a projetos (IPSE) pode ser criado.

4.2.4 Reusabilidade

Trata-se da criação e reuso dos blocos de construção de software. No futuro, seria possível construir um sistema a partir do seu esboço utilizando componentes prontos e com um mínimo de esforço. Atualmente, a realidade ainda não chegou a esse nível, mas continua caminhando nessa direção.

Segundo Pressman (2005, p. 102), se existir um software que cumpra os requisitos, este deverá ser adquirido pois o custo de desenvolvimento com certeza seria maior que o custo de compra. Caso o mesmo necessite de modificações, deve-se agir com cautela, pois o custo de modificação pode ser maior ou muito semelhante ao custo de desenvolvimento de um novo software equivalente.

Esses recursos de software devem ser especificados bem cedo, permitindo uma avaliação técnica de todas as alternativas.

4.3 Estimativas de Projetos de Software

A existência de muitas variáveis, humanas, técnicas, ambientais, dificultam estimativas de custo e de esforço precisas. Segundo Pressman (2005, p. 103), para que estas estimativas sejam confiáveis, pode-se:

a) atrasar a elaboração das estimativas;

b) usar técnicas de decomposição simples para gerar as estimativas;

c) desenvolver um modelo empírico para o custo e esforço;

d) uso de ferramentas de estimativas automatizadas.

A primeira opção não é prática, já as demais são viáveis. A decomposição do projeto facilita a estimativa. O uso de modelos empíricos baseados em dados históricos e que, segundo Pressman (2005), assume a forma de:

$$d = f(v_i) \qquad (10)$$

onde d é o valor a ser estimado e v_i são os parâmetros (por exemplo LOC). As ferramentas de estimativa automatizadas implementam técnicas de decomposição ou modelos empíricos. As estimativas de custo e de esforço dependerão dos dados que o sistema receber. Todas as opções de estimativas sempre dependem de bons dados históricos. Se não houver dados históricos, fica muito mais difícil uma estimativa confiável.

4.4 Estimativas de Linhas de Código (LOC) e Pontos-por-Função (FP)

Para facilitar e possibilitar a estimativa de custo e esforço de um software muito complexo faz-se a decomposição do problema.

Segundo Pressman (2005, p. 104), dados de LOC e FP podem ser usados como variáveis de estimativa, para classificar os elementos do software por tamanho, e como métricas de linha básica (*Baseline*) baseada em dados históricos.

O planejador de software, a partir da declaração de escopo do software, o decompõe em subfunções, estimando então as linhas de código e pontos-por-função para cada uma. Aplicando as métricas de produtividade às variáveis de estimativas, o custo ou esforço é derivado. As estimativas das subfunções são combinadas para obter uma estimativa global. LOC, estimada diretamente, exige maior detalhamento e decomposição, ao ponto que FP, estimada indiretamente, usam dados mais macroscópicos.

Para cada variável de estimativa o planejador irá estimar um valor otimista e um pessimista. Segundo Pressman (2005, p. 105), o valor esperado da estimativa pode ser definido como:

$$E = [(a = 4m + b) / 6] \qquad (11)$$

onde (E) é o valor esperado, (a) é a média ponderada das estimativas otimistas, (m) as mais prováveis e (b) as pessimistas.

Com o valor esperado, aplicam-se os dados de produtividade LOC ou FP. Pressman (2005, p. 106) afirma que o planejador poderá multiplicar o valor de estimativa total para todas as subfunções pela métrica de produtividade média correspondente. Num exemplo de uma estimativa de 310 FP e produtividade de 5,5 FP / pessoas-mês (pm), o esforço global seria:

$$\text{Esforço} = (310 / 5{,}5) = 56 \text{ pm} \qquad (12)$$

Ou valor da variável de estimativa de cada subfunção pode ser multiplicado pelo valor de produtividade ajustado, baseado na complexidade da subfunção. Quanto maior a complexidade, menor será considerada a produtividade, e vice-versa.

As estimativas devem sempre ser corrigidas para refletir mudanças ocorridas. Os resultados obtidos deverão ser cruzados utilizando-se uma técnica diferente para possibilitar uma verificação.

4.5 Estimativa do Esforço

Segundo Pressman (2005, p. 110), esta "é a técnica mais comum para se levantar os custos de qualquer projeto de desenvolvimento de engenharia."

Após o delineamento das funções do software a partir do escopo do projeto, "o planejador estima o esforço (por exemplo, pessoas-mês) que será exigido para se concluir cada tarefa de engenharia de software para cada função de software." (PRESSMAN, 2005, p. 111).

Atribui-se um valor à mão-de-obra para cada tarefa de engenharia de software a ser executada. Os valores obtidos de custo e esforço serão cruzados e comparados com outras estimativas. Pressman (2005) sugere a elaboração de uma matriz do esforço, como a Figura 6.

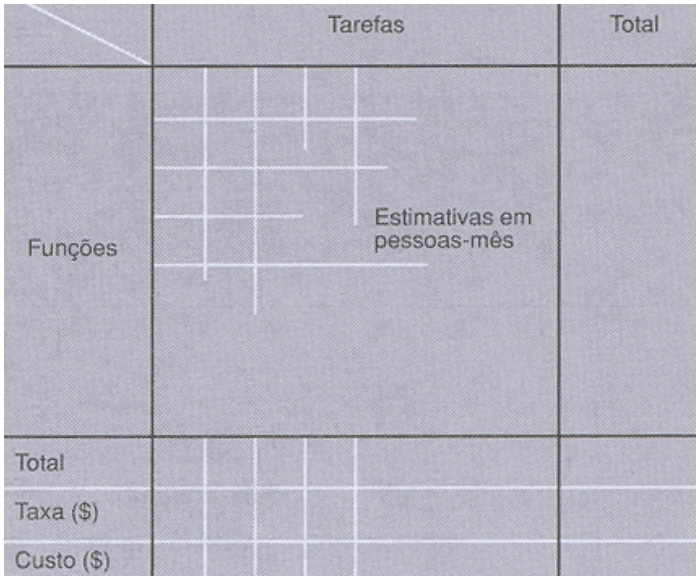

Figura 6: Desenvolvimento de uma matriz do esforço.
Fonte: PRESSMAN, 2005, p. 111.

Para Pressman (2005, p. 113), se ao cruzarmos estimativas distintas encontrarmos divergências muito grandes, as duas causas mais prováveis são a má compreensão ou

interpretação do escopo do projeto ou os dados referentes à produtividade da mão-de-obra são impróprios ou estão obsoletos ou ainda foram mal aplicados. Cabe ao planejador encontrar a causa das divergências e refazer as estimativas.

4.6 Modelos Empíricos de Estimativa

"Os modelos de recursos consistem de uma ou mais equações empiricamente derivadas que prognosticam o esforço (em pessoas-mês), duração do projeto (em meses cronológicos), além de outros dados pertinentes ao projeto." (PRESSMAN, 2005, p. 114).

Basili (*apud* PRESSMAN, 2005, p. 114) descreve quatro classes de modelos de recursos: modelos estáticos de variável simples, modelos estáticos de múltiplas variáveis, modelos dinâmicos de múltiplas variáveis e modelos teóricos. Os dois primeiros modelos assumem a forma de equações simples, já os modelos dinâmicos projetam os requisitos de recursos como uma função do tempo.

Barry Boehm apresenta uma hierarquia de modelos de estimativa de software chamada COCOMO (*Constructive Cost Model*, Modelo de Custo Construtivo). Boehm considera um conjunto de atributos de custo agrupados em quatro grandes categorias: Atributos do produto, do hardware, de pessoal e de projeto. Por meio de um sistema de pontuação, faz-se a estimativa do esforço (PRESSMAN, 2005, p. 115).

O próprio Boehm (*apud* PRESSMAN, 2005, p. 118) teria dito:

> *Atualmente, um modelo de estimativa de custos de software se sai bem se puder estimar os custos de desenvolvimento de software dentro de 20 % dos custos reais, em 70 % do tempo e dentro de seu próprio campo de ação (isto, é dentro das classes de projetos para as quais ele foi calibrado)...Isso não é tão preciso quanto gostaríamos, mas é exato o bastante para nos oferecer alguma ajuda na análise econômica da engenharia de software e nas tomadas de decisão.*

Um exemplo de modelo de estimativa dinâmico é o modelo de Putnam, cuja distribuição de esforço dá origem a um gráfico de curvas de Rayleigh-Norden, como na Figura 7, que indicam o esforço distribuído pelas etapas de desenvolvimento de um software e a tarefa a ser executada.

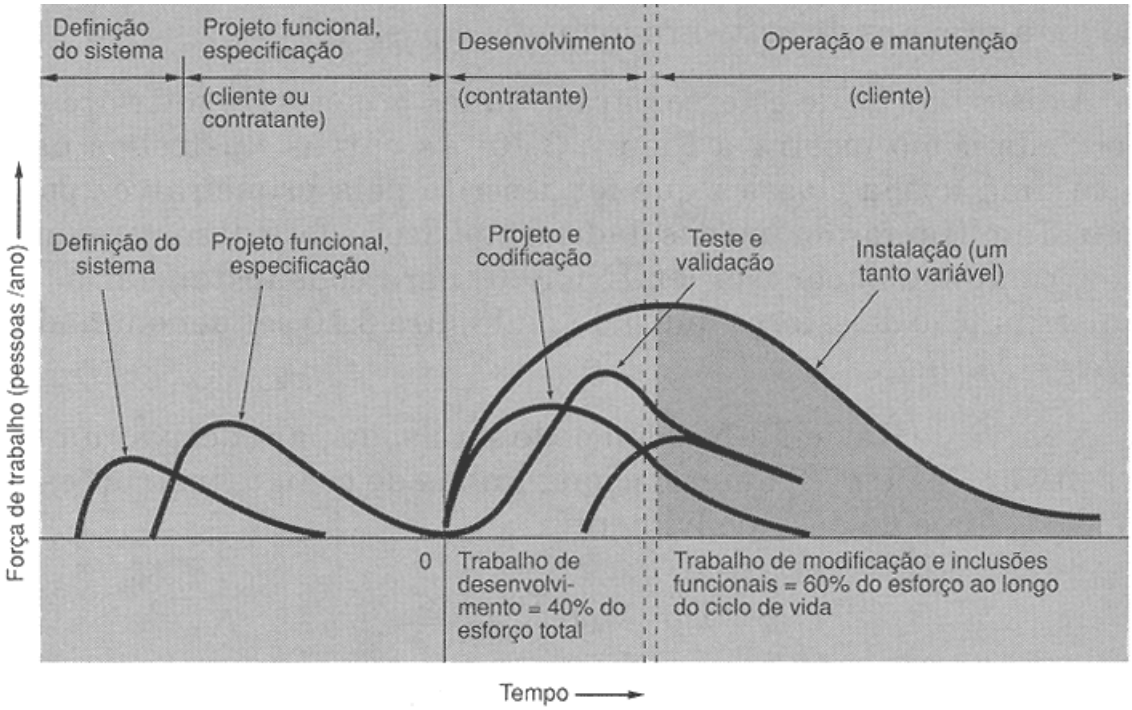

Figura 7: Distribuição do esforço – Grandes fases.
Fonte: PRESSMAN, 2005, p. 120.

Com os estudos de Putnam se chega a uma expressão para o esforço de desenvolvimento K:

$$K = (L^3) / (C_k^3 * t_d^4) \qquad (13)$$

onde K é o esforço (em pessoas-ano), C_k é uma constante que descreve o estado da tecnologia e t_d é o tempo de desenvolvimento em anos. Tanto o modelo COCOMO como o modelo de Putnam se baseiam em estimativas do número de linhas de código.

4.7 Ferramentas de Estimativa Automatizadas

Utilizam as técnicas de decomposição e os modelos de estimativa empíricos, permitindo que o planejador estime custos e esforço e realize a análise de outras importantes variáveis. Essas ferramentas em geral precisam de uma estimativa quantitativa do tamanho do projeto, ou da funcionalidade, de características qualitativas e de alguma descrição do pessoal responsável pelo desenvolvimento. (PRESSMAN, 2005, p. 121)

5 PLANEJAMENTO DE PROJETOS

Mesmo tendo sido feitas todas as estimativas de custo e esforço, antes de começar o desenvolvimento do software propriamente dito, deve-se "responder a algumas questões importantes sobre riscos, desenvolver uma estratégia para atacarmos o problema, estabelecer um mecanismo para avaliarmos o progresso e organizar os membros do pessoal que foram escolhidos para construir o produto." (PRESSMAN, 2005, p. 130).

Quanto a projetos de software, Weinberg (1993, p. 119) afirma:

> *Projetos fogem – explodem ou entram em colapso – porque os gerentes acreditam em uma ou ambas dessas falácias: a falácia reversível (as ações podem ser sempre desfeitas) e a falácia causadora (cada causa tem um efeito, e você pode dizer qual é a causa e qual é o efeito).*

5.1 Análise dos Riscos

Robert Charette (*apud* PRESSMAN, 2005, p. 131) disse que "o risco, assim como a morte e os impostos, é uma das poucas certezas da vida." Durante a análise dos riscos, surge os três pilares de Charette na preocupação com o futuro (riscos que façam o software ser ruim), com a mudança (de tecnologias, nos requisitos do cliente) e com as nossas escolhas (quais métodos e ferramentas usar).

Segundo Pressman (2005, p. 131), "a análise de riscos é, de fato, composta por quatro atividades distintas: identificação, projeção, avaliação e administração dos riscos."

5.1.1 Identificação dos riscos

Alguns riscos são impossíveis de serem identificados antecipadamente, mas há muitos outros que não. A elaboração de uma série de perguntas pode ajudar o planejador a compreender os riscos. Entre as classificações possíveis dos riscos Pressman (2005, p. 132) define:

a) riscos de projeto: problemas orçamentários, de cronograma, pessoal, recursos, clientes e requisitos;

b) riscos técnicos: problemas de projeto, implementação, interface, verificação e manutenção. Ocorrem quando um problema é mais difícil do que o imaginado;

c) riscos do negócio: podem ser subdivididos em outros cinco;

   i) um produto que ninguém quer;

   ii) um produto que não mais se encaixe na estratégia da empresa;

   iii) um produto que não se consegue vender;

   iv) perda do apoio da alta administração;

   v) perda do compromisso orçamentário ou de pessoal.

5.1.2 Projeção dos riscos

A projeção dos riscos tenta determinar a probabilidade de um risco ocorrer e quais seriam as conseqüências se isso acontecesse. Para tanto, deve ser estabelecida uma escala de probabilidade, que poderá ser numérica, baseada, por exemplo, nos dados históricos e medições realizadas, ou ainda qualitativa, com valores que vão de um altamente improvável a um altamente provável (PRESSMAN, 2005, p. 133).

O planejador também deverá delinear as conseqüências dos riscos e sua estimativa de impacto, colocando os riscos em uma ordem de prioridade. O impacto de um risco poderá ser afetado pela sua natureza, o seu escopo, ou seja, o quão sério ele é e quanto ele afetará o projeto, e o tempo de ocorrência.

Por fim deve ser feita uma anotação da precisão da projeção dos riscos. Pressman (2005, p. 135) sugere uma distribuição para o impacto dos riscos e a preocupação gerencial, de acordo com a probabilidade de ocorrerem, como ilustra a Figura 8.

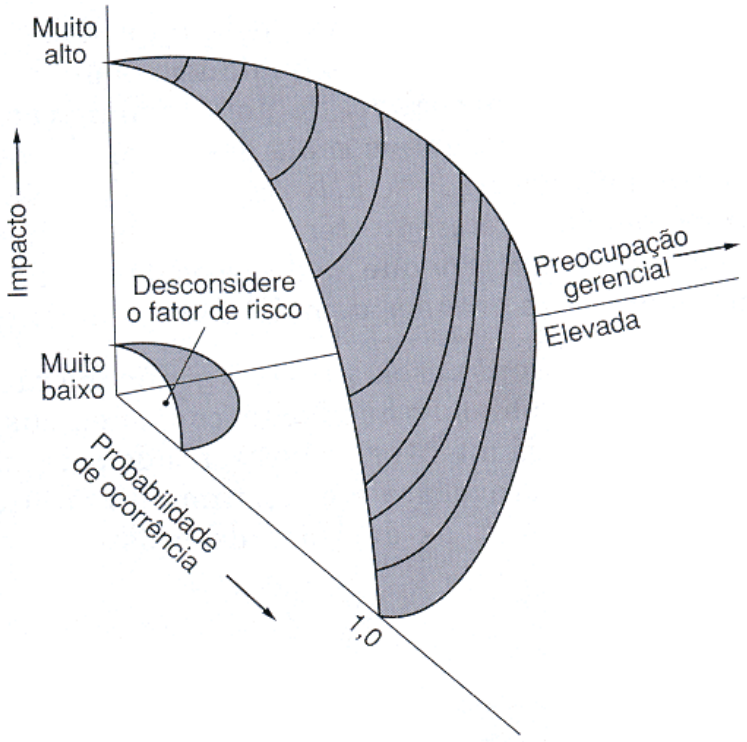

Figura 8: Risco de acordo com o impacto e a probabilidade de ocorrência.
Fonte: PRESSMAN, 2005, p. 135.

5.1.3 Avaliação dos riscos

Weinberg (1993) diz que as pessoas tendem a serem otimistas, acreditando que quando as coisas estão ruins podem melhorar por si mesmas. Este tipo de pensamento deve estar distante durante a avaliação dos riscos. Aqui a precisão das estimativas feitas durante a projeção dos riscos é examinada mais detalhadamente. Também se tenta determinar uma ordem de prioridade para os riscos e encontrar maneiras de evitá-los. Uma relação de três eixos [r_i, l_i, x_i] é estabelecida, como a vista na Figura 8, com r_i sendo o risco, l_i a probabilidade e x_i o impacto. (PRESSMAN, 2005, p. 135)

Os custos, os prazos e os desempenhos terão um nível de risco referente, ou seja, um valor máximo tolerado. O ponto referente ou *break point* será o ponto em que a decisão de prosseguir ou encerrar o projeto têm peso igual.

5.1.4 Gerenciamento e monitoração dos riscos

O gerenciamento dos riscos será desenvolvido baseado na descrição, probabilidade e impacto de cada risco. Com estes conhecidos serão estabelecidos uma série de passos para evitar ou reduzir os riscos. Estes passos geralmente acarretam em custos adicionais, devendo ser determinado o custo-benefício das ações, verificando se realmente é interessante tomá-las.

Em projetos muito grandes, para simplificar o trabalho, Pressman (2005, p. 137) indica o uso da regra 80/20 de Pareto, ou seja, 80% do potencial de fracasso do projeto será correspondente a apenas 20% dos riscos identificados. O planejador deverá identificar esses 20% críticos e trabalhar neles. Todos os dados de administração dos riscos serão organizados num Plano de Administração e Monitoração dos Riscos (PAMR).

Com o PAMR desenvolvido e o projeto iniciado, começa a monitoração dos riscos, onde será avaliado se um risco previsto realmente ocorre, será garantido que os passos de reversão estejam surtindo efeito e serão coletadas informações para o futuro.

5.2 Determinação de um Cronograma

A determinação de um cronograma para projetos de desenvolvimento de software geralmente é feita baseada em uma data final de entrega, distribuindo o esforço dentro do espaço de tempo previsto. O ideal seria que a própria organização pudesse estabelecer a data final.

A elaboração do cronograma leva em consideração todo o planejamento de projeto.

5.2.1 Relações pessoas-trabalho

Num projeto pequeno uma única pessoa pode executar várias tarefas, mas à medida que o projeto aumenta mais pessoas deverão ser envolvidas. O número de pessoas que irão realizar o trabalho deve ser muito bem estimado pois no caso de atraso o acréscimo de novas pessoas pode acarretar em um aumento do atraso, uma vez que as novas pessoas terão de aprender tudo sobre o sistema e quem irá ensiná-los terá de parar com o seu trabalho (PRESSMAN, 2005, p. 142).

Mais pessoas também pode aumentar o número de canais de comunicação, requerendo um esforço adicional e ainda mais tempo. A comunicação para o trabalho em equipe é essencial para melhorar a qualidade e manutenibilidade do software que está sendo desenvolvido.

Pode-se utilizar a equação de Putnam para encontrar a melhor relação entre o número de pessoas envolvidas e o tempo total do projeto.

5.2.2 Definição de tarefas

Quando várias pessoas estão envolvidas em um projeto é provável que algumas atividades sejam desenvolvidas paralelamente. Para tanto o software deverá ter sido bem projetado, com as dependências entre as atividades bem definidas. Na Figura 9 é possível visualizar uma rede de tarefas com a sua seqüência de execução, sendo algumas sendo executadas em paralelo, e alguns marcos de referência que seriam atingidos assim que a documentação produzida como parte de uma tarefa fosse revisada e aprovada.

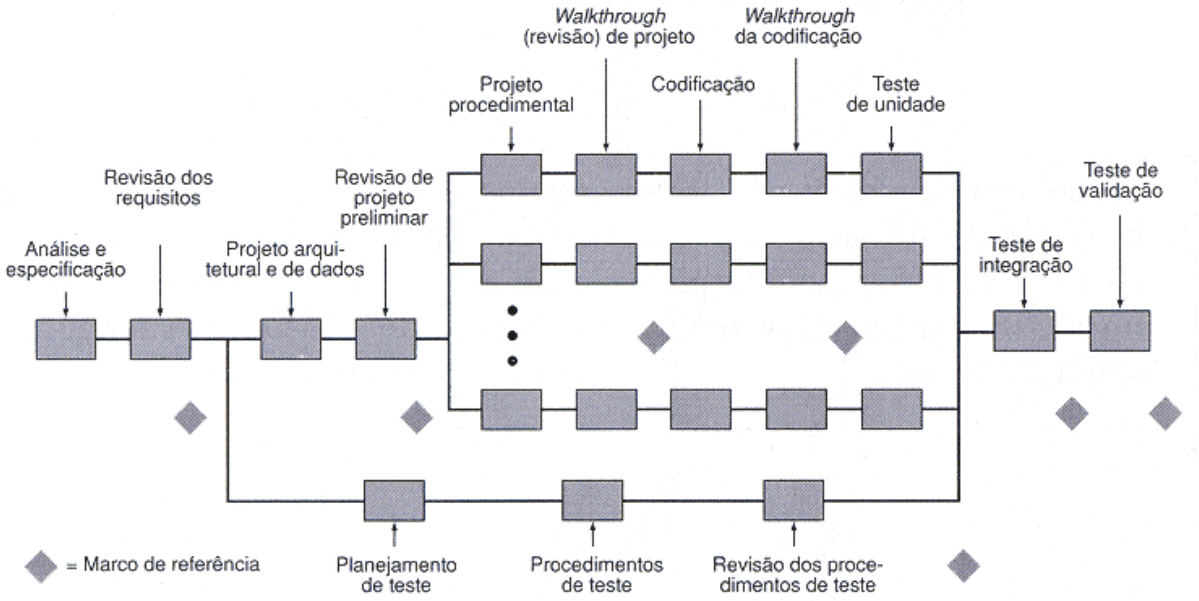

Figura 9: Rede de tarefas e paralelismo.
Fonte: PRESSMAN, 2005, p. 144.

5.2.3 Distribuição do esforço

Segundo Pressman (2005, p. 145), uma distribuição de esforço recomendada segue a proporção de 40-20-40, com 40% do esforço para análise e projeto, 20% para codificação e 40% para testes. Esta proporção deve ser usada apenas como referência, sendo a distribuição feita de acordo com as características de cada projeto. A codificação toma somente 20% do esforço justamente devido ao grande esforço na análise e projeto, facilitando o desenvolvimento do código. No caso de um software *human-rated*, ou seja, cuja falha pode resultar em perdas de vidas humanas, a parte de teste pode demandar mais esforço.

5.2.4 Determinação de cronogramas

Algumas ferramentas e técnicas para determinação de um cronograma de projeto podem ser aplicadas no software com poucas modificações. Tanto o PERT (*Program Evaluation and Review Technique* – método de avaliação e revisão de programa) quanto o CPM (*Critical Path Method* – método do caminho crítico) podem ser aplicados no desenvolvimento de software (PRESSMAN, 2005, p. 146).

A rede de tarefas é definida com o desenvolvimento de uma lista de todas as tarefas e uma lista de disposições, também chamada de lista de restrições. O uso tanto do PERT como do CPM permitem a determinação do caminho crítico, estabelecimento de estimativas de tempo melhores e cálculo de limites de tempo para uma tarefa em particular.

Riggs (*apud* PRESSMAN, 2005, p. 147) descreve cinco limites de tempo importantes, sendo:

a) o limite mais cedo que uma tarefa pode iniciar após todas as que lhe antecedem terem sido encerradas;

b) o limite mais tarde para iniciar a tarefa antes que o tempo mínimo para término do projeto seja atrasado;

c) o término mais breve (soma do início mais cedo e a duração da tarefa);

d) o término mais tardio (soma do início mais tarde e a duração da tarefa);

e) a flutuação total, ou folga para que o caminho crítico seja mantido no prazo.

5.2.5 Rastreamento e controle de projeto

Segundo Pressman (2005, p. 154), o rastreamento (*tracking*) pode ser feito por meio de reuniões com os membros da equipe, avaliando-se os resultados das revisões, verificando se os marcos de referência foram atingidos na data programada, comparando a data de início planejada com a real ou com reuniões informais entre os profissionais.

Segundo Weinberg (1993, p. 139):

> *Para dirigir um projeto de engenharia de software, o gerente precisa:*
> - *Planejar o que deve acontecer.*
> - *Observar que coisas significativas estão realmente acontecendo.*
> - *Comparar o observado com o planejado.*
> - *Tomar ações necessárias para trazer o real próximo do planejado.*

O gerente de projetos deverá efetuar o controle de projeto para poder administrar os recursos, enfrentar problemas e dirigir o pessoal. Se tudo estiver indo bem, este controle será leve. No caso de problemas, o gerente poderá re-alocar pessoas ou redefinir a programação do projeto.

5.3 Adquirir ou Desenvolver o Software?

Às vezes pode ser mais interessante adquirir um software à desenvolvê-lo. No caso de adquirir, geralmente haverá a opção de compra ou licenciamento do software, compra e adaptação ou ainda terceirização do desenvolvimento de um software.

Quando o software for barato, pode ser melhor adquiri-lo e testá-lo a realizar toda uma avaliação do mesmo. Quando este for mais caro, Pressman (2005, p. 160) sugere algumas diretrizes:

a) definir, sempre que possível, características mensuráveis desejadas, como desempenho do software;

b) estimar o custo e data de entrega para o seu desenvolvimento na própria empresa;

c) escolher de três a quatro pacotes que atendam às especificações;

d) comparar suas funções-chave, realizar *benchmark* entre eles;

e) avaliar sua qualidade, suporte, reputação do fornecedor, etc.;

f) pedir opiniões de outros usuários.

O software será adquirido ou não se sua entrega for antes do tempo de desenvolvimento de um software, se o custo da compra e adaptação for menor que o de desenvolvimento, se o custo do suporte externo for menor que o do interno. Para ajudar na decisão podem ser utilizadas técnicas estatísticas, como uma árvore de decisão.

5.4 Reengenharia de Software

A manutenção de programas em fase de envelhecimento pode tornar-se proibitivamente dispendiosa quando um sistema já possui tantos remendos que já não funciona adequadamente. A reengenharia de software pode ser uma alternativa à manutenção, nesses casos. Há uma série de passos que podem ser seguidos para se desenvolver a reengenharia de software.

Primeiramente devem ser selecionados os programas muito usados hoje e que continuarão sendo muito usados num futuro próximo, de 5 a 10 anos. Estima-se o custo da manutenção deste programas e disponha-os em uma ordem de prioridade. Em seguida estima-se o custo da reengenharia, utilizando as técnicas de estimativa, e os custos de manutenção após a reengenharia. Compara-se o custo da manutenção com o da reengenharia e o tempo de retorno do investimento na reengenharia, considerando as vantagens que o sistema terá após a reengenharia, como melhor desempenho, e toma-se a decisão (PRESSMAN, 2005, p. 163).

5.5 Plano de Projeto de Software

“Cada etapa do processo de engenharia de software deve produzir um resultado que possa ser revisado e possa funcionar como base para os passos que se seguirão.” (PRESSMAN, 2005, p. 167).

O plano de projeto de software é um documento simples que combina informações geradas em todas as atividades de estimativa e planejamento. Ele deve indicar o escopo e os recursos do software, definir os riscos e como evitá-los, definir custos e prazos e oferecer uma abordagem global para todos os envolvidos.

6 GARANTIA DE QUALIDADE DE SOFTWARE

A garantia de qualidade de software (*Software Quality Assurance* - SQA) deve ser aplicada durante todo o processo de engenharia de software. Para Pressman (2005, p. 724), a SQA abrange:

a) métodos e ferramentas de análise, projeto, codificação e teste;

b) revisões técnicas formais;

c) uma estratégia de teste;

d) controle da documentação de software;

e) um procedimento para garantir a adequação aos padrões de desenvolvimento de software;

f) mecanismos de medição e divulgação.

É difícil definir se um software é de qualidade. Para isso usaremos uma definição do próprio Pressman (2005, p. 724) onde define qualidade como a "conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo software profissionalmente desenvolvido."

Há controvérsias quanto a esta definição, principalmente por considerar a existência de características implícitas, quando muitos profissionais argumentam que tudo deve ser explicitado. Pressman (2005) considera aqui que esses requisitos implícitos seriam todas as qualidades que sempre se deseja em um software, como a manutenibilidade, e que geralmente não está escrito em lugar nenhum.

6.1 Fatores de Qualidade

Os fatores de qualidade de software podem ser divididos em dois grupos: os que podem ser medidos diretamente, como o número de erros encontrados, e os que podem ser medidos somente indiretamente, como a usabilidade do software.

McCall (*apud* PRESSMAN, 2005, p. 726), enumera uma extensa lista de fatores que afetam a qualidade de software, focalizando no software principalmente suas características operacionais, sua manutenibilidade de mudanças e sua adaptabilidade.

6.2 Atividades de Garantia de Qualidade de Software

A garantia de qualidade de software é um padrão sistemático e planejado de ações. A responsabilidade pela qualidade de software envolve várias pessoas em uma organização, passando pelos engenheiros de software, a equipe de vendas e os clientes (PRESSMAN, 2005, p. 733).

O grupo SQA seria um representante do cliente. Seus membros devem olhar para o software pelo ponto de vista do cliente, observando os fatores de qualidade, os padrões estabelecidos. Pressman (2005, p. 734) associa a qualidade de software a sete grandes atividades:

Aplicação de métodos técnicos: a garantia de qualidade de software inicia desde o projeto, com a escolha de métodos e ferramentas que ajudem a desenvolver um software de alta qualidade.

Realização de revisões técnicas formais: trata-se de um encontro entre o pessoal técnico visando descobrir problemas de qualidade.

Atividades de testes de software: uma série de métodos de projeto de casos de testes que ajudam a garantir uma detecção de erros efetiva.

Aplicação de padrões: os padrões e procedimentos formais, às vezes determinados por clientes ou pela própria organização, exigem uma atividade de SQA para garantir que eles sejam cumpridos.

Controle de mudanças: necessário devido ao grande potencial que as mudanças têm para introduzir novos erros. Deve ser aplicado durante o desenvolvimento e manutenção do software.

Medição: faz parte de qualquer disciplina de engenharia. As medições permitem comparações, determinação de níveis de qualidade e a avaliação do impacto de mudanças.

Manutenção de registros e reportagem: os resultados de revisões, auditorias, controle de mudanças, testes, entre outros, devem compor um registro histórico do projeto.

6.3 Revisões de Software

As revisões são aplicadas em vários pontos durante o desenvolvimento do software e servem para descobrir defeitos que possam ser eliminados. Elas também podem apontar melhorias necessárias. São realizadas durante reuniões de cerca de duas horas de duração, focando-se numa parte específica do software (PRESSMAN, 2005, p. 736).

O grande benefício das revisões é a descoberta precoce dos defeitos. Segundo Pressman (2005, p. 737), um erro descoberto durante a fase de projeto pode ter um custo de correção de uma unidade. Se descoberto pouco antes dos testes, seu custo passa para 6,5 unidades. Durante os testes, 15 unidades e logo após o lançamento do software seu custo de correção variará de 60 a 100 unidades.

Pressman (20005, p. 740) define os cinco principais objetivos das revisões técnicas formais como sendo:

a) descobrir erros;

b) verificar se o software atende aos requisitos;

c) garantir o cumprimento dos padrões pré-definidos;

d) obter um software que seja desenvolvido uniformemente;

e) tornar os projetos mais administráveis.

Cada revisão deve ser documentada com uma lista das questões levantadas, o que foi revisado, quem fez a revisão e quais as descobertas e conclusões.

Como ponto de partida de cada revisão, Pressman (2005) sugere o uso de listas de conferência, as chamadas *checklists*. Pode-se utilizar listas de conferência para a especificação do sistema, para a revisão do plano de projeto de software, revisão da análise de requisitos, revisão do projeto de dados, de arquitetura e procedimental do software, listas para revisão da codificação, revisão do plano e procedimento de teste, para uma revisão voltada à manutenção do software, além de outras *checklists* que podem ser utilizadas durante as revisões técnicas para avaliar outras questões.

6.4 SQA Estatística

A garantia estatística da qualidade reflete uma tendência de quantificar a qualidade de software. Para tanto alguns passos devem ser seguidos, como a coleta de informações sobre os defeitos de software e o rastreamento de suas causas. Utilizando o princípio de Pareto (80% dos defeitos correspondem a 20% de todas as causas possíveis), tenta-se isolar os outros 20%, o chamado “pouco vital” (PRESSMAN, 2005, p. 763).

As informações sobre os defeitos são coletadas e separadas em categorias. Monta-se uma tabela de classificação com o número de ocorrências e sua gravidade. Desta forma determinam-se as causas pouco vitais para iniciar uma ação corretiva sobre elas.

Com todas essas informações calcula-se um índice de defeitos. Este índice pode ser usado para desenvolver uma indicação global da melhoria da qualidade de software. Pressman (2005) resume a aplicação da garantia estatística da qualidade e do princípio de Pareto

dizendo: “gaste seu tempo concentrando-se nas coisas que de fato interessam, mas primeiro certifique-se de entender o que realmente interessa!”

6.5 O Processo Sala Limpa

O processo Sala Limpa, ou *cleanroom*, visa a prevenção de defeitos, ou seja, encontrar os defeitos antes mesmo de se iniciar os testes. Segundo Pfleeger (2004, p. 354), o processo busca “certificar o software com relação às especificações, em vez de esperar pelo teste de unidades para encontrar os defeitos.”

O processo utiliza uma verificação matemática e testes estatísticos do uso. Cobb e Mills (*apud* Pfleeger, 2004, p. 356), relatam que estes testes são 20 vezes mais efetivos do que os testes tradicionais.

Experiências com o processo Sala Limpa mostram que se obtiveram ótimos resultados para equipes de desenvolvimento pequenas, cerca de 10 pessoas com projetos também pequenos, até cerca de 50000 linhas de código, com ganho de produtividade e grande redução de defeitos. Para projetos maiores, no entanto, ele não se mostrou eficaz.

Contra o uso do processo Sala Limpa, Beizer (*apud* PFLEEGER, 2004, p. 357), considera que não se pode encontrar um erro a menos que se execute o código com este erro, e o processo Sala Limpa realiza somente testes estatísticos para verificar a confiabilidade do software, evitando qualquer tipo de teste de unidade.

6.6 Implantando a Garantia de Qualidade de Software

É comum haver resistência à implantação da garantia de qualidade de software por considerar que isto apenas trará novos custos e mais burocracia e que já é feito tudo que poderia ser feito para se ter um software de qualidade. (PRESSMAN, 2005, p. 775)

O primeiro passo para que se possam implantar os procedimentos de garantia de qualidade é a realização de uma auditoria para que se determine qual o estado atual da garantia de qualidade

de software e do gerenciamento da configuração de software. Feito isto é possível concluir se há a necessidade de uma SQA ou não.

Segundo Pressman (2005, p. 776), a garantia de qualidade de software pode trazer os seguintes benefícios:

a) software com menos defeitos, resultando em redução do esforço e tempo gasto com manutenção e testes;

b) uma maior confiabilidade que resultará em maior satisfação do cliente;

c) redução dos custos de manutenção.

Em contrapartida, sua implantação pode ser difícil de ser aplicada em pequenas organizações, devido aos poucos recursos e por representar uma mudança cultural, o que nunca é fácil. Se os custos dos erros que podem ocorrer sem a implantação da garantia de qualidade de software forem superiores à soma dos custos de implantação com o novo custo de erros, então a garantia de qualidade de software é efetiva.

7 TESTE DE SOFTWARE

Segundo Deutsch (*apud* PRESSMAN, 2005, p. 786):

> *O desenvolvimento de sistemas de software envolve uma série de atividades de produção em que as oportunidades de injeção de falhas humanas são enormes. Erros podem começar a acontecer logo no começo do processo, onde os objetivos... podem estar errônea ou imperfeitamente especificados, além de erros que venham a ocorrer em fases de projetos e desenvolvimento posteriores... Por causa da incapacidade que os seres humanos têm de executar e comunicar com perfeição, o desenvolvimento de software é acompanhado por uma atividade de garantia de qualidade.*

"A atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificação, projeto e codificação." (PRESSMAN, 2005, p. 786).

Os testes de software são executados na tentativa de identificar os defeitos de software que possam provocar a sua falha, antes que o software chegue até o cliente. Segundo Pfleeger (2004), pode-se dizer que um software falhou quando ele não faz o que os requisitos descrevem, sendo a falha o resultado de um ou mais defeitos. Para Pfleeger (2004, p. 270), a falha pode ter várias razões:

a) especificação errada ou falta de um requisito;

b) especificação com um requisito impossível de ser implementado, por qualquer razão;

c) defeito no projeto;

d) erro no código.

7.1 Classificando os Defeitos

Os diversos defeitos que podem ocorrer num software podem ser classificados de acordo com a natureza do defeito. Segundo Pfleeger (2004, p. 272 e p. 273), os defeitos podem ser assim classificados:

Defeitos de sintaxe: uso inadequado dos princípios da linguagem utilizada, como a falta de uma vírgula. Atualmente, os compiladores são capazes de encontrar a maioria dos defeitos de sintaxe.

Defeitos de computação e de precisão: quando uma fórmula está errada o seu resultado não retorna com a precisão necessária.

Defeitos na documentação: quando a documentação não está de acordo com o que o programa realmente faz.

Defeitos por sobrecarga ou *stress*: quando o software é extrapolado além do seu limite máximo. Quando as estruturas de dados recebem mais dados do que o especificado.

Defeitos relativos à capacidade: quando o desempenho do sistema se torna inaceitável à medida que a atividade do sistema alcança um limite especificado.

Defeitos de sincronia: quando não ocorre a coordenação dos diversos processos que devem ser executados simultaneamente ou em uma seqüência.

Defeitos de desempenho: quando o sistema não funciona na velocidade determinada pelos requisitos.

Defeitos de recuperação: quando, após uma falha ter ocorrido, o sistema se comporta diferentemente do esperado.

Defeitos do hardware e do software: quando o hardware ou software com que o sistema trabalha não se comporta do modo esperado.

Defeitos de padrões e procedimentos: quando os padrões não são seguidos, defeitos podem ser gerados à medida que o sistema é testado e modificado.

Há várias outras classificações dos tipos de defeitos. A própria IBM e também a Hewlett-Packard criaram as suas. Classificar e analisar os tipos de defeitos encontrados pode ajudar a prever os tipos de defeitos que o software poderá ter, ajudando no direcionamento dos testes.

7.2 Teste de Software

É comum que os testes representem 40% do esforço total de um projeto. No caso de softwares de sistemas vitais, dos quais dependam vidas humanas, os testes podem custar de três a cinco vezes mais que todos os outros passos da engenharia de software juntos. Porém, quando um erro é descoberto com o software já na mão do usuário, o custo da correção poderá ser de 60 a 100 vezes maior do que durante a fase de desenvolvimento (PRESSMAN, 2005, p. 786).

Para Glen Myers (*apud* PRESSMAN, 2005, p. 788) a intenção da atividade de teste é descobrir erros. Um bom caso de teste é aquele com grande probabilidade de encontrar erros e um teste bem-sucedido é aquele que revela um erro ainda não conhecido. Uma atividade de teste não pode provar que não existam erros no sistema, até porque dependendo da complexidade do mesmo a ausência de erros é algo que provavelmente nunca deixará de ser uma mera utopia.

Os resultados obtidos pelos testes de software são comparados com os resultados esperados. Quando se encontram erros inicia-se a depuração. O processo de depuração é totalmente imprevisível, pois nunca se sabe quanto tempo levará para se corrigir um erro, podendo ser horas ou até semanas.

Se durante os testes são encontrados erros graves, que exijam modificação de projeto, significa que a qualidade e confiabilidade do software são suspeitas e novos testes devem ser realizados. Caso contrário, pode-se concluir que o software apresenta qualidade e confiabilidade adequadas, ou que os testes estão sendo inadequados para encontrar os erros mais graves.

7.2.1 Planejamento do teste

Pfleeger (2004, p. 299) considera que o processo de teste possui vida própria dentro do ciclo de vida de desenvolvimento podendo ser realizado em paralelo com outras atividades. O planejamento do teste incluiria as seguintes etapas:

a) estabelecimento dos objetivos do teste, informando que tipos de casos de teste gerar;

b) projeto dos casos de teste, essencial para o restante do processo de testes;

c) elaboração dos casos de teste que serão utilizados;

d) execução dos casos de testes;

e) avaliação dos resultados dos testes.

O teste começaria com a revisão dos casos de teste, verificando sua corretitude, viabilidade e se cobrem as funcionalidades pretendidas.

7.2.2 Organização dos testes

O teste geralmente envolve vários estágios. Segundo Pfleeger (2004, p.275) os seguintes testes podem ser implementados:

a) teste de módulo: cada componente do programa é testado isolado dos outros componentes do sistema;

b) teste de integração: verifica se os componentes, quando juntos, trabalham de acordo com a especificação;

c) teste funcional: determina se as funções estão mesmo sendo realizadas;

d) teste de desempenho: compara o sistema com o restante dos requisitos de software e hardware;

e) teste de aceitação: feito junto ao cliente, compara-se o software com os requisitos do cliente;

f) teste de instalação: testa a instalação no ambiente em que será utilizado.

7.2.3 Projeto de casos de teste

O projeto de casos de teste de software deve ter o objetivo de garantir que os testes tenham a maior probabilidade de encontrar a maioria dos erros com o mínimo de tempo e esforço.

Para Pressman (2004, p. 791), qualquer produto de engenharia pode ser testado de duas formas: conhecendo-se a função específica que um produto deve executar, realizando testes que demonstrem que cada função é totalmente operacional, ou ainda conhecendo-se o funcionamento interno de um produto, realizando testes que garantem que a operação interna do produto tem um desempenho de acordo com as especificações e que os componentes internos foram adequadamente postos à prova.

A primeira abordagem é chamada de teste de caixa preta (*black box*), realizado nas interfaces do software, e a segunda é chamada de teste de caixa branca (*white box*), que se baseia em um minucioso exame dos detalhes procedimentais.

7.3 Teste de Caixa Branca

O teste de caixa branca, que geralmente é executado cedo no processo de teste, testa os caminhos lógicos do software. O programa pode ser examinado em vários pontos e verificado se o comportamento deste é de acordo com o esperado. Um teste de caixa branca completo, definindo-se todos os caminhos lógicos, poderia garantir um software sem falhas, porém o tempo necessário para a sua realização cresce exponencialmente e mesmo em pequenos programas a estimativa de tempo pode passar das centenas de anos.

Usando os métodos de caixa branca o engenheiro de software pode garantir que todos os caminhos de um módulo sejam exercidos ao menos uma vez, que exercitem as decisões lógicas para valores falsos e verdadeiros, que executem todos os laços em suas fronteiras e dentro de seus limites operacionais e exercitem as estruturas de dados internas para garantir a sua validade (PRESSMAN, 20005, p. 793).

Segundo Pressman (2005, p. 794), o teste de caixa branca é interessante por três fatores principais: 1) erros lógicos e pressuposições são inversamente proporcionais à probabilidade

de que um caminho de programa seja executado. 2) um caminho lógico muitas vezes pode ser executado regularmente mesmo quando se pensa o contrário. 3) erros de digitação podem acontecer.

Estes tipos de erros geralmente são muito difíceis de serem verificados em testes de caixa preta.

7.4 Teste de Caixa Preta

Os testes de caixa preta são projetados para validar os requisitos funcionais, sem se preocupar com o funcionamento interno de um programa. Para Pressman (2005, p. 816), deve ser utilizado como uma abordagem complementar ao teste de caixa branca, podendo encontrar uma classe de erros diferente.

O teste de caixa preta visa encontrar erros em funções incorretas ou ausentes, erros de interface, erros nas estruturas de dados, erros de desempenho e erros de inicialização e término.

O teste de caixa preta tende a ser aplicado nas últimas etapas das atividades de teste. Segundo Pressman (2005, p. 816), os testes devem ser projetados para responderem às seguintes perguntas:

a) como a validade funcional é testada?

b) quais classes de entrada serão bons casos de teste?

c) o sistema é mais sensível a determinados valores de entrada?

d) como são isoladas as fronteiras de uma classe de dados?

e) quais índices e volumes de dados o sistema pode tolerar?

f) qual o efeito de combinações específicas de dados na operação do sistema?

7.5 Ferramentas Automatizadas de Teste

Existem muitas ferramentas automatizadas que ajudam nos testes dos componentes de código, com destaque para as ferramentas para análise de código, para execução de testes e geradores de casos de teste (PFLEEGER, 2004, p. 300).

7.5.1 Ferramentas para análise de código

Dividem-se em análise estática, realizada quando o programa não está sendo executado, e análise dinâmica, quando o programa está sendo executado (PFLEEGER, 2004, p. 300).

As ferramentas de análise estática podem, segundo Pfleeger (2004), ser divididas em quatro grupos:

a) analisador de código: avalia a sintaxe dos componentes;

b) verificadores de estrutura: gera um grafo do fluxo de dados, verificando problemas estruturais;

c) analisador de dados: analisa as estruturas de dados;

d) verificador de seqüência: verifica a seqüência dos eventos.

No caso de análise dinâmica, as ferramentas possibilitam a visualização dos estados dos eventos durante a execução do programa, podendo gerar estatísticas dos valores assumidos por variáveis e do número de vezes que um componente é chamado.

7.5.2 Ferramentas para execução de testes

“Devido ao tamanho e à complexidade da maioria dos sistemas atuais, as ferramentas automatizadas para a execução de testes são essenciais para lidar com números muito grandes de casos de teste...” (PFLEEGER, 2004, p. 302).

As ferramentas de captura e repetição capturam as teclas digitadas, as entradas e as respostas, e comparam com a saída que é esperada. Várias ferramentas podem ser integradas com outras, formando um amplo ambiente de testes.

Pfleeger (2004, p. 302) alerta que as ferramentas automatizadas, por mais que elas possam ajudar, encontram as evidências de um defeito, o que não é o mesmo de localizar o defeito. "O teste sempre envolverá o esforço manual requerido para rastrear um problema até sua causa original. A automação auxilia, mas não necessariamente substitui essa função humana."

7.5.3 Geradores de casos de teste

O teste depende da boa definição do caso de teste, sendo interessante automatizar parte deste processo para garantir que os casos de teste cobrirão todas as situações possíveis.

Além de geradores de casos de teste estruturais, que baseiam seus casos de teste na estrutura do código-fonte, os geradores podem ter como base o fluxo de dados, o teste funcional ou ainda o estado das variáveis.

7.6 Quando Encerrar os Testes

Às vezes, pode-se levar muito tempo para que se descubram defeitos triviais. À medida que aumenta o número de defeitos encontrados logo no início, cresce também a probabilidade da existência de mais defeitos não detectados, o que pode contrariar um pouco a nossa intuição. Isso torna muito difícil decidir quando parar os testes (PFLEEGER, 2004, p. 303).

Para poder interromper os testes é preciso estimar o número de defeitos remanescentes, podendo assim ter um certo grau de confiança no software.

Harlan Mills desenvolveu a técnica de implantação de defeitos, onde um membro da equipe de testes insere intencionalmente um número de defeitos no software. Após os testes verificam-se quantos defeitos inseridos foram detectados e toma-se a proporção de defeitos inseridos e detectados em relação aos ainda não detectados como um parâmetro para os

defeitos em geral. Quando esta proporção atingisse um valor aceitável de confiança no software, os testes seriam dados por encerrados (PFLEEGER, 2004, p. 303).

Pressman (2005, p. 842) também sugere o uso de um modelo logarítmico proposto por Musa e Ackerman que permite prever a diminuição de erros à medida que a atividade de teste progride. A intensidade de erros real pode ser traçada em relação à curva logarítmica, como pode ser visto na Figura 10, permitindo prever a quantidade de testes necessários para se atingir uma intensidade de falhas baixa.

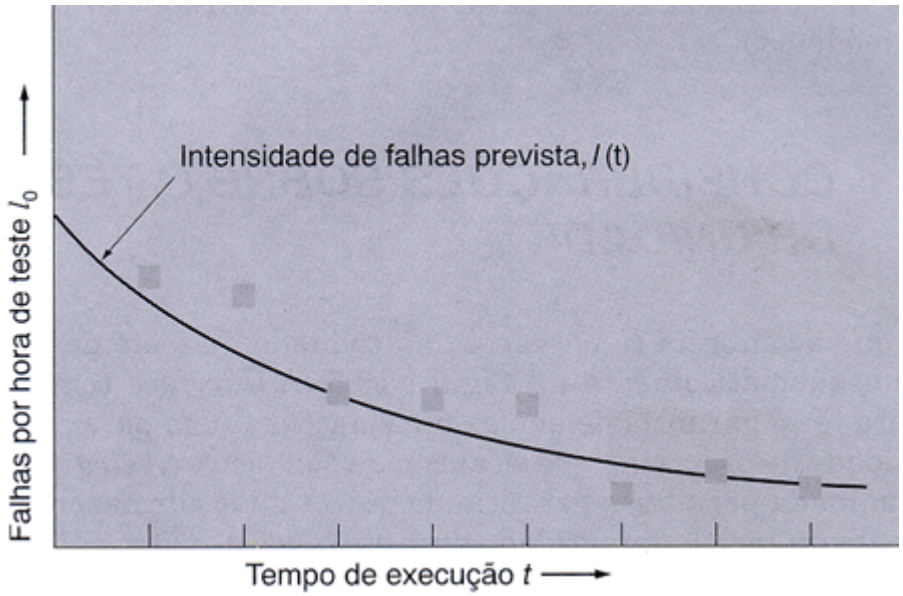

Figura 10: Intensidade de falhas como uma função do tempo de execução.
Fonte: PRESSMAN, 2005, p. 843.

No caso das ferramentas automatizadas, pode-se usar como referência o número de caminhos percorridos e testes executados. Há ainda a possibilidade de simplesmente testar o sistema até atingir uma determinada data no cronograma, o que geralmente ocorre quando se trabalha com prazos muito apertados, ou quando não houver mais dinheiro, podendo em ambos os casos resultar em softwares com muitos defeitos não detectados.

Pressman (2005, p. 830) afirma ainda que as atividades de teste nunca terminam, sendo apenas transferidas para o cliente, que toda vez que usa o programa está também realizando um teste. Cabe ao engenheiro de software descobrir e corrigir o maior número de erros possível antes que os testes do seu cliente se iniciem.

8 ENTREGANDO E MANTENDO O SOFTWARE

Após todo o processo de desenvolvimento do software chega o momento de entregá-lo ao cliente. A respeito disso, Pfleeger (2004, p. 366) diz:

> *... a entrega envolve mais do que simplesmente instalar o sistema no local de operação. Esse é o momento do desenvolvimento em que ajudamos os usuários a entenderem e a se sentirem mais à vontade com nosso produto. Se a entrega não for bem-sucedida, os usuários não utilizarão o sistema adequadamente e poderão ficar descontentes com o seu desempenho. De qualquer modo, os usuários não serão tão produtivos ou eficientes como poderiam ser e nosso trabalho de construir um sistema de alta qualidade terá sido em vão.*

Uma vez entregue, o software precisará de manutenção com o passar do tempo, afinal o cliente não pode ser abandonado simplesmente pelo fato de o software ter sido entregue. Novos erros poderão ser encontrados e novas funcionalidades poderão ser agregadas nesse software. A manutenção do software possui algumas semelhanças com a manutenção de equipamentos, porém o software não se desgasta nem necessita de peças novas, o que não significa que se a manutenção do mesmo não for realizada ele não deixará de atender ao cliente.

8.1 Treinamento

Pfleeger (2004, p. 366) considera que um sistema é utilizado por duas pessoas: o usuário, que utiliza as funções principais do sistema, e o operador, que realiza funções auxiliares, como *backups*, instalação, recuperação de arquivos danificados, entre outras. Por terem funções diferentes, Pfleeger (2004) propõe um treinamento diferente para cada um. Às vezes, ambos são uma única pessoa.

O treinamento do usuário foca nas principais funções do sistema, o que elas fazem, como ele irá acessá-las e qual o procedimento para utilizá-las. O usuário não precisa saber sobre as operações internas do sistema.

O treinamento do operador foca nas funções de suporte do sistema, abordando como o sistema funciona e o que ele faz. O operador é treinado a operar o novo sistema e também a prestar suporte técnico aos usuários.

O treinamento de usuários e operadores começa com os aspectos básicos do sistema, durante a entrega do software. Outros treinamentos podem ser marcados posteriormente, tanto para reciclagem como para novos usuários e operadores. Há também a possibilidade de realização de treinamentos especiais, focados em uma determinada função que é importante para um grupo de pessoas, enquanto um outro grupo recebe treinamento em outra função que seja interessante para eles.

O treinamento pode se dar em forma de aula ou um curso, ao vivo ou pela Internet. Para facilitar o aprendizado é importante que o sistema seja projetado com uma interface clara e fácil de entender, com ícones e ajuda acessíveis.

8.2 Documentação

A documentação de um software é muito importante não só no treinamento como para o seu sucesso. Um sistema de computador pode ser utilizado por várias pessoas, desde usuários comuns, operadores, profissionais de suporte a sistemas, entre outros. Cada documento poderá conter informações diferentes para cada tipo de pessoa (PFLEEGER, 2004, p. 370).

Os manuais do usuário são um guia de referência para os usuários do sistema. Ele deve ser completo e compreensível, abrangendo o propósito do sistema, suas capacidades e funções, os recursos, características e vantagens do sistema, além de uma descrição de todas as suas funções.

O manual do operador é muito semelhante ao do usuário, porém traz outras informações como o desempenho do sistema, configurações de hardware e software, gerenciamento de usuários, modo de realizar *backup*, entre outras que não são abrangidas pelo manual do usuário. Uma ajuda (*help*) na tela do sistema também é muito importante tanto para operadores como para os usuários, como uma fonte rápida de informação e de alcance fácil.

A documentação pode incluir ainda um guia geral do sistema, que permita ao cliente saber de forma rápida o que o sistema faz, bem como tutoriais que o ensinem passo a passo, além de muitos outros documentos que podem ser entregues, como um documento de requisitos, um guia do programador, entre outros.

8.3 Manutenção de Software

Segundo Pressman (2005, p. 876), a manutenção de software pode representar até 70% de todo o esforço despendido por uma organização de software. De fato, pode-se chegar à situação em que a empresa não consegue mais produzir nada novo porque está gastando todos os seus recursos para manter os softwares antigos.

Pressman (2005, p. 877) divide a manutenção de software em quatro atividades que se assemelham muito à classificação da manutenção de equipamentos:

a) manutenção corretiva: correção de erros encontrados após a entrega do software;

b) manutenção adaptativa: modifica o software devido às mudanças tecnológicas que ocorrem com o passar do tempo;

c) manutenção perfectiva: responsável pela maior parte de todo o esforço despendido na manutenção de software, compreende as modificações em funções existentes e acréscimos de outras em função dos pedidos dos usuários;

d) manutenção preventiva: quando o software é modificado para melhorar a confiabilidade ou a manutenibilidade futura.

No caso da manutenção adaptativa e perfectiva ocorrem as mesmas tarefas aplicadas durante a fase de desenvolvimento, determinando novos requisitos, reprojetando, gerando código e testando.

8.4 Manutenibilidade

Segundo Pressman (2005, p. 884), “a manutenibilidade pode ser definida qualitativamente como a facilidade com que um software pode ser entendido, corrigido, adaptado e/ou aumentado.” A manutenibilidade “orienta os passos de um processo de engenharia de um software.”

A manutenibilidade de um software seria afetada por diversos fatores, desde negligência na fase de projeto até uma configuração de hardware ruim. Pressman (2005) cita uma série de fatores que pode estar relacionada ao ambiente de desenvolvimento:

a) pessoal de software qualificado;

b) estrutura de sistema compreensível;

c) facilidade de manuseio;

d) linguagens de programação padronizadas;

e) sistemas operacionais padronizados;

f) estrutura de documentação padronizada;

g) disponibilidade de casos de teste;

h) facilidades de depuração;

i) um computador adequado para a manutenção.

Ainda segundo Pressman (2005, p. 885), “possivelmente, o fator mais importante a afetar a manutenibilidade seja o planejamento para a manutenibilidade.”

8.4.1 Métricas da manutenibilidade

Assim como a qualidade e a confiabilidade, a manutenibilidade pode ser um pouco difícil de ser medida, porém Gilb (*apud* PRESSMAN, 2005, p. 886) propõe uma série de métricas indiretas relacionadas ao esforço necessário durante a manutenção:

a) tempo de reconhecimento do problema;

b) tempo de retardo administrativo;

c) tempo de coleta de ferramentas de manutenção;

d) tempo de análise do problema;

e) tempo de especificação das mudanças;

f) tempo de correção (ou modificação) ativa;

g) tempo de testes locais;

h) tempo de testes globais;

i) tempo de revisão de manutenção;

j) tempo de recuperação total.

Além destas medidas, a complexidade lógica e estrutura de programa também influem na manutenibilidade do software.

8.5 Efetuando a Manutenção

Pressman (2005, p. 887) considera que a manutenção de software começa muito antes que um pedido de manutenção seja feito, devido à organização e planejamento que as tarefas de manutenção demandam. Não é necessário estabelecer uma organização formal de manutenção, a menos que o software seja muito grande, mas uma delegação informal de responsabilidades seria totalmente essencial.

Essa delegação de responsabilidades reduziria a confusão e melhoraria o fluxo das atividades de manutenção. Ela também estabeleceria uma área de responsabilidade pela manutenção.

8.5.1 Pedidos de manutenção

Os pedidos de manutenção feitos pelo cliente devem ser apresentados de forma padronizada. Pressman (2005, p. 889) sugere a elaboração de um formulário com todas as informações sobre o erro que foi encontrado e as circunstâncias em que foi encontrado ou uma breve especificação da mudança desejada, no caso de manutenção adaptativa ou perfectiva.

Internamente, a organização deverá avaliar o esforço exigido pelo pedido, a natureza das modificações e sua prioridade.

8.5.2 Seqüência de eventos

Pressman (2005, p. 890) sugere uma seqüência de eventos ilustrada na Figura 11 como resultado de um pedido de manutenção.

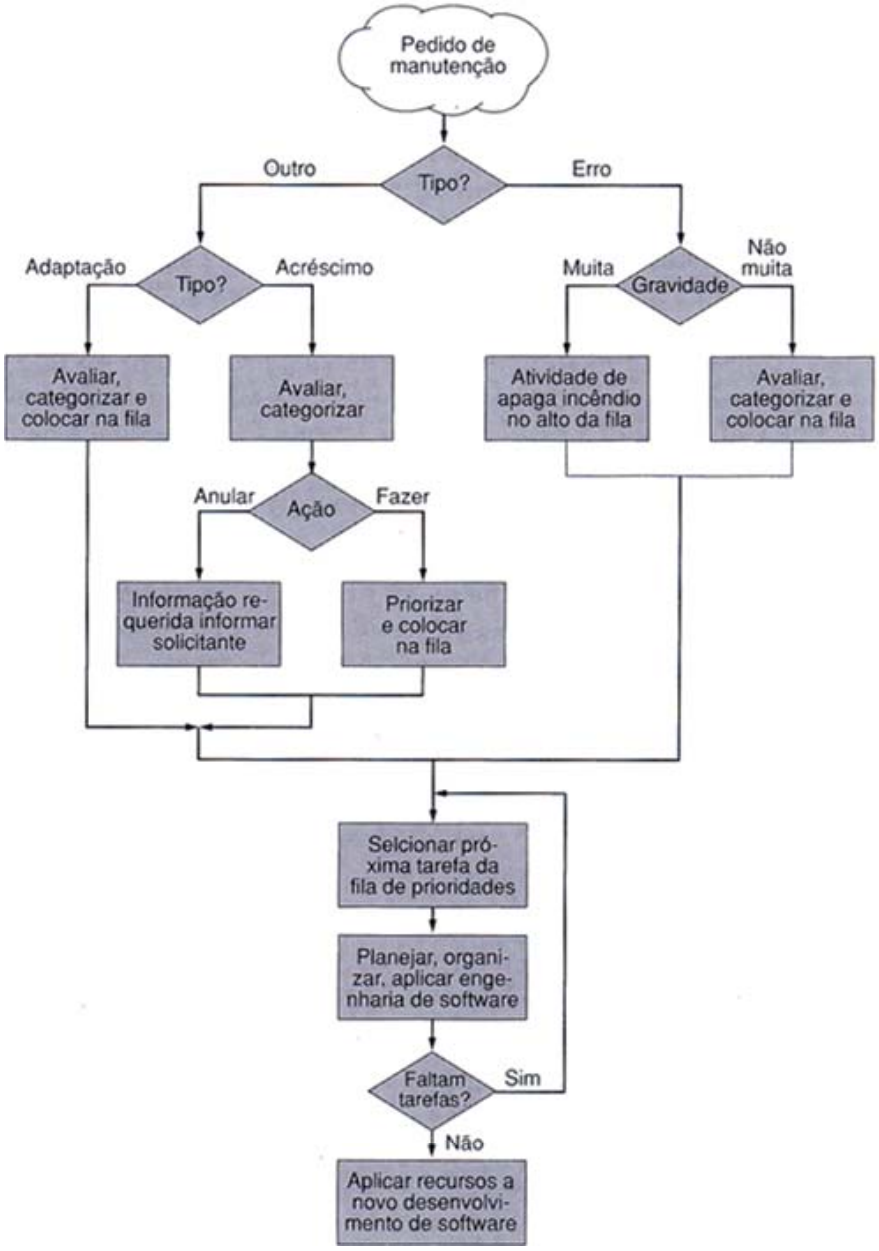

Figura 11: Fluxo de eventos de manutenção.
Fonte: PRESSMAN, 2005, p. 891.

É interessante observar a atividade de apaga incêndio, que ocorre em situações de crise quando há um erro tão grave que o código deve ser modificado imediatamente.

Em todos os casos deve ocorrer a modificação do projeto de software, revisões, modificações do código, testes, caracterizando a manutenção de software realmente como uma atividade de engenharia de software.

Depois de terminada a tarefa de manutenção, Pressman (2005) sugere uma revisão da situação para verificar o que poderia ter sido feito diferente que facilitaria a manutenção, fornecendo informações para esforços de manutenção futuros.

8.5.3 Avaliação da manutenção

Para que se possa avaliar a efetividade das técnicas de manutenção é necessário que sejam guardados registros da manutenção que está ocorrendo. Com estes registros é possível desenvolver uma série de medidas de manutenção. Swanson (*apud* PRESSMAN, 2005, p. 895) apresenta uma lista de medidas potenciais, como o número médio de falhas, total de pessoas empregadas em mudanças e correções, percentagem de pedidos de manutenção por tipo, entre outras medidas possíveis.

Estas medidas formam uma estrutura de dados quantitativos que podem ser utilizados para analisar a técnica de desenvolvimento empregada, projeções sobre o esforço de manutenção, alocação de recursos e outras questões além da possibilidade de avaliar a própria tarefa de manutenção.

8.6 Efeitos Colaterais da Manutenção

Tratam-se de erros ou comportamentos indesejáveis que ocorram como resultado de uma modificação. Os efeitos colaterais poderiam ser classificados em três categorias: Efeitos colaterais na codificação, nos dados e na documentação (PRESSMAN, 2005, p. 896).

Os efeitos colaterais na codificação ocorrem da modificação do código. Podem variar de pequenos erros, como um rótulo escrito erradamente, até a erros catastróficos que façam o software falhar durante a execução.

Os efeitos colaterais nos dados ocorrem como resultado de modificações feitas na estrutura de informações do software. Uma documentação de projeto que descreva a estrutura de dados e forneça uma referência cruzada que associe elementos de dados, registros, arquivos e outras estruturas aos módulos de software pode contribuir para limitar os efeitos colaterais.

Os efeitos colaterais na documentação ocorrem quando há mudanças no software que não são refletidas na documentação do projeto nem nos manuais do usuário. Para Pressman (2005, p. 898), “uma documentação de projeto que não reflita precisamente o estado atual do software provavelmente é pior do que nenhuma documentação.” Isto porque novos efeitos colaterais poderão ocorrer no futuro se novos esforços de manutenção forem feitos baseados em uma documentação incorreta.

9 ESTUDO DE CASO DA EMPRESA THEÒS INFORMÁTICA LTDA.

9.1 Histórico da Empresa

A Theòs Informática Ltda. é uma pequena empresa que nasceu da necessidade de um padre da cidade de Dourados, no estado de Mato Grosso do Sul, que precisava de um software para facilitar e agilizar o trabalho de sua secretaria paroquial. Enxergando aí uma boa oportunidade, dois irmãos, sendo um deles padre canonista, fundaram a empresa em 1997 com o objetivo de desenvolver o software que atenderia às necessidades deste e de vários outros padres da Igreja Católica do Brasil.

A empresa desenvolveu três softwares específicos para a Igreja. O SGCP, Sistema de Gestão Canônico-Pastoral, com versões específicas para paróquias, cúrias e seminários. Devido ao número de paróquias ser muito maior do que o número de cúrias e de seminários, a empresa tem focado a maior parte de seus esforços no desenvolvimento do SGCP-Paróquia.

Nascida em Dourados, a empresa mudou-se para Maringá em julho de 1999. Até hoje ela se concentra no desenvolvimento, venda e no suporte de seus três softwares para Igreja. Atualmente, a Theòs Informática Ltda. está comercializando o SGCP-Paróquia versão 4.1, o SGCP-Cúria versão 4.1 e o SGCP-Seminário versão 2.0.

9.2 Caracterização da Empresa

A Theòs Informática é uma empresa que atua no desenvolvimento, venda e suporte de software voltado especificamente para a Igreja Católica. Atuando no mercado desde 1997, seus produtos compreendem três softwares: O SGCP-Paróquia, SGCP-Cúria e SGCP-Seminário. A empresa ainda é pequena, com apenas seis funcionários atualmente, e sua sede está localizada em Maringá desde 1999.

No período de 2002 a 2005 a empresa atuou também na área de venda e manutenção de equipamentos, mas hoje se concentra somente em sua linha de softwares SGCP.

9.3 Metodologia

O estudo de caso consistiu na análise de um questionário entregue aos clientes da empresa Theòs Informática e, com o uso destas informações e das ferramentas e metodologias aqui apresentadas, foi dado início ao projeto da nova versão do SGCP-Paróquia, com a elaboração de um cronograma das atividades, estimativas de recursos necessários e de custos.

9.4 Análise do SGCP-Paróquia

Com o objetivo de desenvolver uma nova versão de seu software SGCP-Paróquia, a Theòs Informática Ltda. elaborou um formulário de análise que foi entregue para mais de 1000 paróquias que adquiriram o SGCP-Paróquia nos últimos anos. Os itens deste questionário foram propostos de modo a abranger todos os recursos do sistema, da forma mais concisa possível para que o mesmo não se tornasse muito extenso.

Com as informações deste formulário a empresa obteve um retorno de seus clientes quanto ao software, seus pontos fortes e fracos e onde deveria melhorar. A partir do diagnóstico realizado com os clientes e da experiência já adquirida a empresa definiu as especificações da nova versão do SGCP-Paróquia.

A análise do SGCP-Paróquia por parte dos clientes vem ao encontro da garantia de qualidade de software, em que uma de suas premissas é que o software deve ser visto do ponto de vista do cliente.

9.4.1 Descrição do SGCP-Paróquia

O SGCP-Paróquia é um software desenvolvido para paróquias da Igreja Católica. Seu intuito é facilitar e agilizar todos os trabalhos executados pela secretaria paroquial, com cadastro de fiéis, processos sacramentais (Batismo, Eucaristia, Crisma, Matrimônio), oferta de dízimos, catequese, receitas e despesas da paróquia, planejamento de atividades e cursos, além de informatizar os documentos como livros de registros e certidões antigos que a paróquia já possuir.

Até a data de primeiro de novembro de 2006, a Theòs Informática já havia comercializado 1556 sistemas SGCP-Paróquia para todo o Brasil. No entanto, sabe-se que muitas das primeiras paróquias que adquiriram o SGCP-Paróquia atualmente utilizam outros sistemas, ou simplesmente abandonaram a informatização de sua paróquia e não estão utilizando nenhum software específico, ou ainda outras situações desconhecidas. Atualmente estima-se em aproximadamente 1000 paróquias que estão de fato fazendo uso do SGCP-Paróquia.

9.4.2 Composição do formulário de análise

O formulário (Apêndice A) entregue às paróquias foi desenvolvido com foco na usabilidade de todas as funções do sistema, valorizando o ponto de vista do usuário, uma vez que é difícil para um leigo dar informações precisas sobre corretitude, manutenibilidade, integridade. O formulário é composto por dezenove questões fechadas onde o cliente deverá dar uma nota de 1 a 5, sendo 1 a nota mais baixa e 5 a nota mais alta. O fato de serem questões fechadas facilita a análise dos itens dos formulários após estes serem preenchidos, permitindo a geração de dados estatísticos.

Todas as questões possuem um espaço aberto para que o cliente possa anotar qualquer sugestão ou reclamação em relação ao item que está sendo perguntado. O formulário de análise também encoraja o cliente a anexar documentos que ele considere relevantes ou que ajudem no entendimento de alguma observação que ele tenha feito.

Algumas questões foram elaboradas especificamente para os usuários das versões mais recentes do software, pois são a respeito dos últimos recursos adicionados no mesmo. Os usuários que possuírem versões sem tal recurso são orientados a deixarem tais questões em branco.

9.4.3 Análise das questões fechadas

Apesar de terem sido enviados aproximadamente 1000 formulários de análise para seus clientes, a empresa recebeu apenas 82 formulários preenchidos. Portanto, para uma população de 1000 obteve-se uma amostra de 82, o que garante um nível de confiança aceitável. No entanto, a margem de erro não foi calculada.

Todos os formulários foram analisados e tiveram suas respostas agrupadas por questão e pela nota dada. O Quadro 1 exibe os totais percentuais das notas recebidas de todas as respostas fechadas.

Pelo Quadro 1 é possível perceber que muitos usuários deixaram algumas questões sem resposta. Isto se explica pelo fato de que muitas destas questões foram deixadas em branco por serem a respeito de itens do SGCP-Paróquia que os usuários simplesmente não estavam utilizando, o que foi salientado pelos mesmos nos campos de observação.

Percentuais	Nota 5	Nota 4	Nota 3	Nota 2	Nota 1	Sem nota
Questão 1	39,02%	36,59%	23,17%	0,00%	0,00%	1,22%
Questão 2	43,90%	31,71%	18,29%	1,22%	1,22%	3,66%
Questão 3	34,15%	39,02%	13,41%	10,98%	0,00%	2,44%
Questão 4	39,02%	23,17%	15,85%	7,32%	3,66%	10,98%
Questão 5	39,02%	29,27%	12,20%	3,66%	0,00%	15,85%
Questão 6	26,83%	15,85%	2,44%	2,44%	0,00%	52,44%
Questão 7	26,83%	8,54%	6,10%	2,44%	1,22%	54,88%
Questão 8	31,71%	23,17%	7,32%	3,66%	2,44%	31,71%
Questão 9	53,66%	24,39%	7,32%	1,22%	0,00%	13,41%
Questão 10	48,78%	30,49%	7,32%	0,00%	0,00%	13,41%
Questão 11	39,02%	26,83%	4,88%	1,22%	2,44%	25,61%
Questão 12	35,37%	25,61%	9,76%	0,00%	2,44%	26,83%
Questão 13	41,46%	12,20%	8,54%	4,88%	1,22%	31,71%
Questão 14	32,93%	18,29%	9,76%	2,44%	4,88%	31,71%
Questão 15	36,59%	18,29%	9,76%	1,22%	2,44%	31,71%
Questão 16	19,51%	25,61%	7,32%	4,88%	1,22%	41,46%
Questão 17	44,44%	24,69%	2,47%	3,70%	1,23%	23,46%
Questão 18	31,71%	28,05%	8,54%	1,22%	0,00%	30,49%
Questão 19	35,37%	24,39%	10,98%	1,22%	0,00%	28,05%

Quadro 1: Percentuais de respostas das questões.

Pelo Quadro 1 observa-se que um grande número de usuários não deu nota às questões 6, relativa à quantidade de informações da ficha Censo, e 7, relativa à facilidade de uso deste.

A razão disso surge ao se analisar o Quadro 2. Essas duas questões, como pode ser observado no questionário do Apêndice A, deveriam ser respondidas apenas por aqueles que possuíssem o SGCP-Paróquia versão 4.1, pois o Censo é um recurso presente somente nesta versão.

Como visto pelo Quadro 2, 33 usuários ou 40,24% dos que responderam ao questionário possuíam versões inferiores. Desta forma conclui-se que esta é a principal razão pelo elevado número de formulários sem nota atribuída às questões do Censo, fato observado no Quadro 1.

Versão	Usuários	
4.1	49	59,76%
4.0	16	19,51%
3.9/3.91	14	17,07%
3.8	2	2,44%
3.6	1	1,22%

Quadro 2: Versão utilizada pelos usuários que responderam aos questionários.

A Figura 12 ilustra graficamente a proporção de utilização de cada versão do SGCP-Paróquia na amostra analisada.

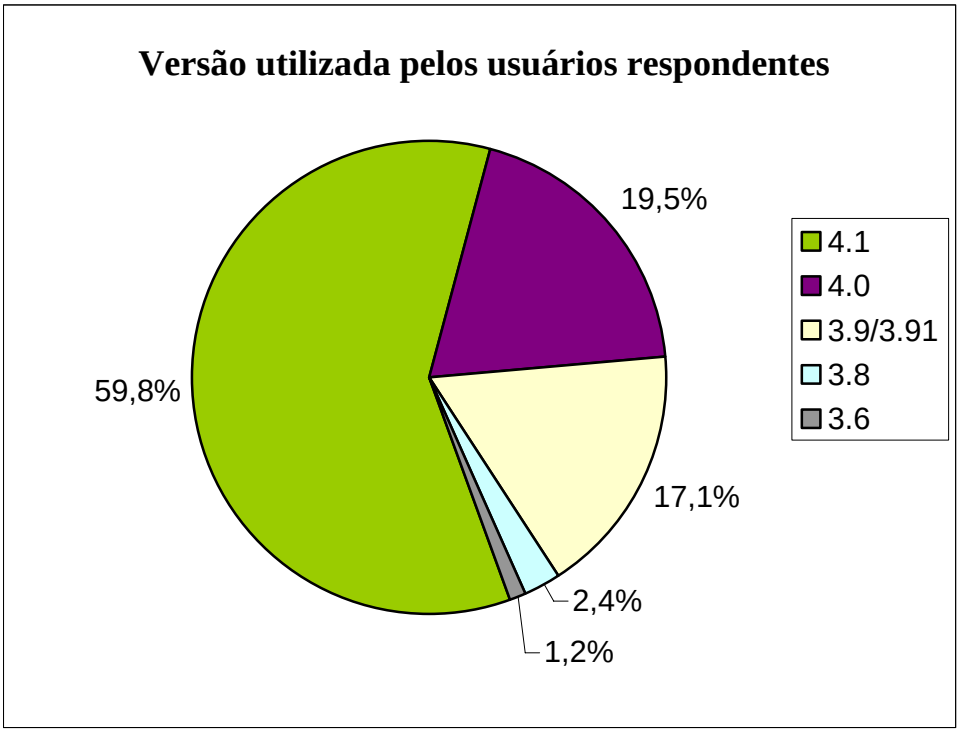

Figura 12: Percentual de usuários respondentes por versão do SGCP.

Em alguns casos mesmo usuários que possuíam a versão mais recente do SGCP-Paróquia deixaram algumas questões em branco. Ao analisar os questionários e as observações feitas nos mesmos percebeu-se que a questão sem nota geralmente significava que a pessoa não utilizava aquele recurso do programa. Obviamente houve também casos em que as questões em branco não possuíam nenhuma justificativa, podendo significar, por exemplo, que a

pessoa que o preencheu simplesmente esqueceu de atribuir uma nota ao item ou deliberadamente não o quis fazer.

Para verificar as respostas somente das questões que receberam notas, montou-se o Quadro 3, onde foram recalculados os percentuais desconsiderando todas as questões que não receberam notas. Assim obteve-se uma informação mais condizente por contabilizar somente as respostas direcionadas aos usuários que utilizam os recursos pesquisados.

Percentuais	Nota 5	Nota 4	Nota 3	Nota 2	Nota 1	Média	Desvio Padrão
Questão 1	39,51%	37,04%	23,46%	0,00%	0,00%	4,16	0,782
Questão 2	45,57%	32,91%	18,99%	1,27%	1,27%	4,20	0,883
Questão 3	35,00%	40,00%	13,75%	11,25%	0,00%	3,99	0,974
Questão 4	43,84%	26,03%	17,81%	8,22%	4,11%	3,97	1,154
Questão 5	46,38%	34,78%	14,49%	4,35%	0,00%	4,23	0,860
Questão 6	56,41%	33,33%	5,13%	5,13%	0,00%	4,41	0,818
Questão 7	59,46%	18,92%	13,51%	5,41%	2,70%	4,27	1,071
Questão 8	46,43%	33,93%	10,71%	5,36%	3,57%	4,14	1,052
Questão 9	61,97%	28,17%	8,45%	1,41%	0,00%	4,51	0,715
Questão 10	56,34%	35,21%	8,45%	0,00%	0,00%	4,48	0,652
Questão 11	52,46%	36,07%	6,56%	1,64%	3,28%	4,33	0,926
Questão 12	48,33%	35,00%	13,33%	0,00%	3,33%	4,25	0,932
Questão 13	60,71%	17,86%	12,50%	7,14%	1,79%	4,29	1,057
Questão 14	48,21%	26,79%	14,29%	3,57%	7,14%	4,05	1,197
Questão 15	53,57%	26,79%	14,29%	1,79%	3,57%	4,25	1,014
Questão 16	33,33%	43,75%	12,50%	8,33%	2,08%	3,98	1,000
Questão 17	58,06%	32,26%	3,23%	4,84%	1,61%	4,46	0,997
Questão 18	45,61%	40,35%	12,28%	1,75%	0,00%	4,30	0,755
Questão 19	49,15%	33,90%	15,25%	1,69%	0,00%	4,31	0,793

Quadro 3: Percentuais das notas das respostas válidas, com sua média e desvio padrão.

Ao olhar os valores do Quadro 3 percebe-se que vários itens obtiveram um boa avaliação pelos usuários do SGCP-Paróquia, com notas variando de 4 a 5, resultando numa média alta. O desvio padrão nos auxilia a observar a dispersão dos valores das notas. Quanto maior o desvio padrão, maior foi a dispersão. Ao mesmo tempo nota-se que houve itens que não receberam uma avaliação tão boa assim, como a questão 3, referente à facilidade de uso do sistema. Esta situação fica bem visível no Quadro 4, que exibe a soma das avaliações com notas 4 ou 5 em ordem decrescente.

O Quadro 4 permite visualizar quais são os itens mais bem avaliados pelos usuários e também os itens que não são tão bem avaliados por eles. Por exemplo, o fato das questões 9 e 10, que se referem ao cadastro de Batismo e aos seus relatórios, respectivamente, terem ótimas avaliações evidencia que o Batismo é um ponto forte do software, bem como a parte do dízimo e seus relatórios, que corresponde à questão 17. Por outro lado, a questão 4, referente à completude de informações da ficha de fiéis, a questão 3, referente à facilidade de uso do SGCP como um todo, e a questão 14, que diz respeito à facilidade de uso do Processo Matrimonial apenas, obtiveram uma má avaliação, significando que os usuários não estão muito satisfeitos com estes itens do programa. Fazendo um paralelo com o Quadro 3 observa-se que a questão 14, que obteve o maior valor de desvio padrão, realmente possui suas notas dispersadas entre todas as faixas de valor.

	Decrescente		**Crescente**		**Crescente**	
Ordem	**Nota 5 ou 4**		**Nota 3**		**Nota 2 ou 1**	
1	**Q 10**	91,55%	**Q 17**	3,23%	**Q 01**	0,00%
2	**Q 17**	90,32%	**Q 06**	5,13%	**Q 10**	0,00%
3	**Q 09**	90,14%	**Q 11**	6,56%	**Q 09**	1,41%
4	**Q 06**	89,74%	**Q 09**	8,45%	**Q 19**	1,69%
5	**Q 11**	88,52%	**Q 10**	8,45%	**Q 18**	1,75%
6	**Q 18**	85,96%	**Q 08**	10,71%	**Q 02**	2,53%
7	**Q 12**	83,33%	**Q 18**	12,28%	**Q 12**	3,33%
8	**Q 19**	83,05%	**Q 13**	12,50%	**Q 05**	4,35%
9	**Q 05**	81,16%	**Q 16**	12,50%	**Q 11**	4,92%
10	**Q 08**	80,36%	**Q 12**	13,33%	**Q 06**	5,13%
11	**Q 15**	80,36%	**Q 07**	13,51%	**Q 15**	5,36%
12	**Q 13**	78,57%	**Q 03**	13,75%	**Q 17**	6,45%
13	**Q 02**	78,48%	**Q 14**	14,29%	**Q 07**	8,11%
14	**Q 07**	78,38%	**Q 15**	14,29%	**Q 08**	8,93%
15	**Q 16**	77,08%	**Q 05**	14,49%	**Q 13**	8,93%
16	**Q 01**	76,54%	**Q 19**	15,25%	**Q 16**	10,42%
17	**Q 03**	75,00%	**Q 04**	17,81%	**Q 14**	10,71%
18	**Q 14**	75,00%	**Q 02**	18,99%	**Q 03**	11,25%
19	**Q 04**	69,86%	**Q 01**	23,46%	**Q 04**	12,33%

Quadro 4: Percentuais de respostas com notas 4 ou 5, nota 3 e notas 2 ou 1.

Um outro aspecto interessante é notar que algumas funcionalidades do sistema receberam notas boas e ruins, ao mesmo tempo. Como exemplo há o Censo, que recebeu uma boa nota com a questão 6, referente à quantidade de informações de sua ficha, porém recebeu uma

avaliação inferior na questão 7, que se refere à sua facilidade de uso e recebeu um bom número de notas 3, 2 ou 1.

Essas conclusões podem ser reforçadas pelos dados ilustrados na Figura 13, que traz a média das notas de todas as questões, já apresentada no Quadro 3. Os mesmos dados estão no Quadro 5, que apresenta a ordem decrescente da média das notas. A ordem decrescente ajuda a definir os itens mais bem avaliados do SGCP.

Mais uma vez observa-se que as questões 9, 10 e 17, referentes à tela de cadastro de batizados, aos relatórios do Batismo e aos relatórios do dízimo respectivamente, obtiveram as melhores médias entre todas. Na outra ponta do quadro, mesmo que em ordens diferentes, as mesmas questões do Quadro 4 são novamente as piores pela avaliação dos usuários que responderam ao questionário.

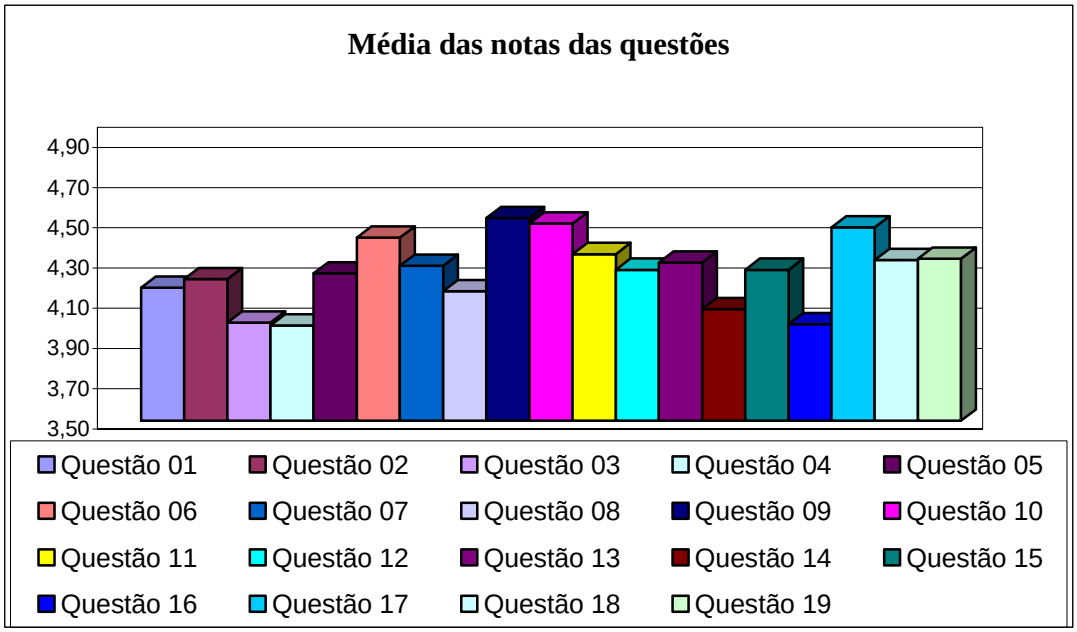

Figura 13: Gráfico da média das notas dadas pelos usuários.

	Nota média
Questão 9	4,51
Questão 10	4,48
Questão 17	4,46
Questão 6	4,41
Questão 11	4,33
Questão 19	4,31
Questão 18	4,30
Questão 13	4,29
Questão 7	4,27
Questão 12	4,25
Questão 15	4,25
Questão 5	4,23
Questão 2	4,20
Questão 1	4,16
Questão 8	4,14
Questão 14	4,05
Questão 3	3,99
Questão 16	3,98
Questão 4	3,97

Quadro 5: Média das notas das questões, em ordem decrescente.

9.4.4 Análise das sugestões

Após o diagnóstico realizado através das notas aplicadas pelos usuários, seguiu-se a análise das sugestões ou observações acrescentadas por eles. Estas observações permitiram entender melhor os motivos que levaram às notas baixas em alguns itens e notas altas em outros, ou até mesmo a ausência de notas. Notou-se que:

a) Muitos usuários não conhecem bem o programa, pois fizeram sugestões de recursos que já existem e inclusive fazem parte da versão que eles utilizam;

b) Muitas paróquias utilizam apenas uma parte do programa, deixando de usar vários recursos interessantes, ou seja, efetuando uma informatização incompleta;

c) Um pedido recorrente foi o de novos relatórios ou de alteração dos já existentes. Alguns usuários anexaram modelos de como eles gostariam que estes fossem;

d) Outra questão comum foi a dificuldade encontrada por grande parte dos usuários no uso de recursos mais complexos, como os itens relacionados ao financeiro e também ao matrimônio, sendo este último bastante extenso devido às exigências do Código de Direito Canônico, ou ainda operações que envolvam arquivos externos, como anexar fotos aos cadastros;

e) Uma questão interessante levantada por alguns usuários é que estes gostariam de cadastrar os livros de registros em folhas de frente e verso, mesmo que ao serem impressas ficassem em folhas separadas. Isso porque os livros reais possuem frente e verso e ao serem passadas as informações para o computador o número da folha não podia ser alterado por se tratar de um livro de registros;

f) Muitos usuários levantaram a possibilidade de uma maior integração entre as partes de Catequese, Crisma e Primeira Eucaristia, que já são integrados, mas deveriam receber alguns ajustes;

g) Houve reclamações da falta de uma interface com a Internet, principalmente no que diz respeito aos e-mails dos fiéis;

h) O recurso de cópia de segurança, apesar de essencial, não é utilizado por muitas paróquias. Às vezes por desconhecimento de sua importância, ou até mesmo pela ignorância da existência do recurso;

i) Alguns itens, em especial o cadastro de fiéis e o censo, obtiveram notas bastante diferentes de cada usuário. Enquanto alguns, por exemplo, consideraram o cadastro de fiéis bastante completo, atribuindo-lhe por isso nota 5, outros o consideraram com dados em excesso, atribuindo-lhe nota 1. Isto fica evidente pelo alto valor do desvio padrão.

Todas as sugestões foram analisadas em grupo, onde foi discutido como o SGCP-Paróquia poderia melhorar os seus pontos fracos, sem prejudicar os pontos fortes, e quais as sugestões viáveis de serem implementadas. Após a análise foram tiradas algumas conclusões úteis para o desenvolvimento da nova versão do SGCP-Paróquia.

9.5 Conclusões da Análise do SGCP-Paróquia

Pelos formulários entregues puderam-se fazer algumas conclusões importantes em relação ao SGCP-Paróquia:

a) O programa é, em geral, considerado pelo usuário como sendo bom, bastante completo e atendendo suas necessidades;

b) Muitos usuários gostariam que o SGCP-Paróquia fosse mais fácil de utilizar;

c) A maioria não utiliza todos os recursos do programa;

d) O recurso mais utilizado, e também mais bem avaliado, é a parte referente ao Batismo;

e) Outro recurso também bastante utilizado e bem avaliado refere-se ao Dízimo;

f) Alguns usuários gostariam de novos relatórios, alguns bastante específicos, ou de poder alterar os já existentes;

g) A ajuda do programa e o manual que o acompanham não são utilizados por uma boa parcela dos usuários;

h) Algumas paróquias gostariam de mais recursos de segurança quanto ao controle de acesso de usuários;

i) A possibilidade de cadastro de livros com folhas de verso é um pedido bastante recorrente;

j) O cadastro de fiéis deve ser atualizado para acrescentar endereço de e-mail.

k) O assistente de cópia de segurança deve ser modificado para que seja mais fácil e seguro, incentivando o seu uso pelos usuários.

Levando em conta todos estes dados deu-se início ao projeto da nova versão do SGCP-Paróquia.

9.6 Elaborando uma nova versão do SGCP-Paróquia

Com os dados dos questionários e a própria experiência no contato com os usuários do SGCP-Paróquia, a Theòs Informática começou a trabalhar na elaboração de uma nova versão do seu programa.

Na determinação de como seria a nova versão, as principais preocupações foram:

a) Produzir um programa melhor que a atual versão;

b) As boas qualidades do programa devem ser mantidas;

c) A nova versão deve ser mais fácil de usar, mais intuitiva;

d) Uma vez que é impossível atender a todos os pedidos de relatórios, o programa deverá possuir um modo que permita a criação e alteração dos mesmos;

e) A segurança quanto ao acesso dos usuários deve ser maior, com mais opções nos níveis de acesso;

f) O programa deve ser rápido, estável e confiável;

g) Como a atual versão do SGCP-Paróquia faz uso de um banco de dados do Microsoft Access, vários problemas de incompatibilidade de versões surgem quando o usuário possui instalado em seu computador uma versão diferente daquela que foi utilizada no desenvolvimento do SGCP. Para evitar tais incompatibilidades, a nova versão não fará mais uso de um banco de dados do Microsoft Access;

h) O SGCP-Paróquia será acompanhado de um novo manual completo e passo-a-passo de todos os recursos, com linguagem clara que incentive sua leitura. O mesmo vale para a ajuda interna do sistema.

Seguindo esta linha desenvolveu-se o projeto da nova versão do SGCP com novos recursos para atender aos pedidos dos usuários das atuais versões e visando a facilidade de acesso aos dados. Destacam-se:

a) Um novo banco de dados, mais robusto, leve e ajustado para o uso em redes, com recursos que garantam a confiabilidade dos dados;

b) Novo tratamento de usuários;

c) Um verdadeiro construtor de relatórios, onde o usuário poderá criar os relatórios que precisa e que o programa não possui;

d) Novos modos de busca que facilitem o acesso a registros quando os dados disponíveis estão incompletos;

e) Maior integração da Catequese com a Crisma, incluindo a Primeira Eucaristia, e uma reformulação para torná-la mais intuitiva;

f) Remodelação do item Censo, tornando-o mais intuitivo e integrado com o cadastro de fiéis;

g) Remodelação do item Matrimônio para que fique mais simples e intuitivo, porém mantendo sua fidelidade com o Código de Direito Canônico e normas da CNBB;

h) Remodelação do modo de trabalho com arquivos externos, como fotos, possibilitando que estes possam ser utilizados independente da sua localização no computador e facilitando o seu uso;

i) Inclusão de e-mails nos cadastros.

9.7 Desenvolvendo o projeto do novo SGCP-Paróquia

Para iniciar o projeto da nova versão do sistema SGCP-Paróquia foram seguidas as etapas propostas neste trabalho. As especificações do novo software foram determinadas pelas sugestões recebidas tanto de clientes como dos próprios funcionários e também do resultado da análise dos formulários de avaliação.

9.7.1 Determinação do escopo

O escopo do software foi definido como sendo um software para paróquias da Igreja Católica que visa facilitar e agilizar todos os trabalhos executados pela secretaria paroquial. O software deverá ser confiável e estável. Para facilitar na determinação das estimativas, as funções do software foram divididas.

9.7.2 Estimativa de recursos

Verificou-se a necessidade de pessoas com habilidades em programação e criação de relatórios, utilizando-se as ferramentas *Delphi 2005* para o desenvolvimento e *Report Builder 10* para os relatórios. Durante o desenvolvimento do projeto será utilizado também o sistema *MS Project*, para acompanhamento de projetos.

Serão empregadas duas pessoas no desenvolvimento, mais uma para os testes e uma quarta pessoa para a documentação. Para o trabalho serão necessários computadores com boa capacidade de processamento e memória, sendo um para cada desenvolvedor. Uma outra máquina será usada para as atividades de teste, sendo esta com a configuração básica de mercado para simular o computador do usuário, mais um quarto computador para a elaboração da documentação.

Durante o desenvolvimento a reusabilidade será intensa, visto que vários elementos poderão ser utilizados em mais de uma parte do software, como por exemplo a interface de busca padrão, presente em praticamente todo o sistema, é desenvolvida somente uma vez, poupando muito tempo de trabalho.

9.7.3 Estimativa do esforço

A estimativa de esforço foi elaborada com base na experiência dos programadores envolvidos. Uma vez delineadas as funções do software estimou-se o número de horas para a conclusão de cada uma. Atribuindo-se um valor à mão-de-obra, chegou-se também a uma estimativa de custo.

9.7.4 Análise dos riscos

Antes de iniciar o desenvolvimento foi feita uma análise dos riscos deste projeto, onde se procurou identificar os riscos "pouco vitais", seguindo Pareto. Assim foram identificados os riscos de tecnologia, riscos de falta de recursos e os riscos humanos.

Para minimizar os riscos de tecnologia escolheram-se ferramentas consagradas no mercado, conceituadas e conhecidas pela comunidade desenvolvedora, estáveis, seguras, de bom desempenho e com grande quantidade de material escrito sobre as mesmas. Foram escolhidos para o desenvolvimento o sistema *Borland Delphi 2005* aliado ao construtor de relatórios *Report Builder 10* e ao banco de dados *Firebird 1.5*.

Quanto aos riscos da falta de recursos, a empresa Theòs Informática manteve uma reserva de caixa específica para o andamento do projeto. Esta reserva corresponde ao custo estimado do projeto mais uma margem de segurança.

Por serem poucas pessoas envolvidas, os maiores riscos são justamente de essas pessoas saírem da empresa durante o projeto. Devido a isto todos da empresa deverão estar cientes, na medida do possível, do andamento do mesmo, o que está sendo feito, para que uma eventual troca de pessoa seja absorvida da forma mais rápida possível.

9.7.5 Determinação de um cronograma

A determinação do cronograma não foi feita em função de uma data de entrega, mas sim de acordo com as estimativas de esforço. É evidente que quanto antes o projeto estiver concluído melhor será.
Por se tratar de uma pequena empresa, com poucas pessoas envolvidas no projeto, a função de rastreamento e controle de projeto fica bastante facilitada. A administração dos recursos e uma eventual re-alocação de pessoal é muito mais simples e fácil de comunicar a todos pelo fato de que todos os envolvidos estão sempre presentes no mesmo ambiente.

Com base nas funções delineadas e nos tempos necessários para implementá-las, com o pessoal disponível, sempre primando por um software de alta qualidade, foi elaborado o

cronograma e o gráfico de Gantt de desenvolvimento do SGCP-Paróquia 5.0, que podem ser vistos no Apêndice B.

Pelo cronograma, atribuindo-se um valor estimado de R$ 20,00 por hora por profissional. As atividades de determinação do escopo, garantia de qualidade de software, análise de requisitos e projeto têm duração estimada em 395 horas de trabalho somadas, sendo necessários duas pessoas. O desenvolvimento e codificação em si, 908 horas. Os testes, 376 horas e a documentação 480 horas de trabalho, cada uma ocupando uma pessoa. Desta forma, tem-se o seguinte cálculo:

$$(395 * 2 * 20{,}00) + (908 * 2 * 20{,}00) + (376 * 20{,}00) + (480 * 20{,}00) = 69240{,}00$$

Portanto o projeto terá um custo estimado total de R$ 69240,00 num total de 2159 horas de trabalho. Esta estimativa exclui custos indiretos, como a compra de ferramentas e equipamentos, pois estes já foram adquiridos. Também não se considera gastos com instalações, aluguel, Luz elétrica, transporte, férias, encargos, entre outros, pois foi considerado que a maioria destes custos, ainda que com alguma variação de valores, já existiria mesmo que a empresa não fosse implementar o projeto do novo SGCP-Paróquia. Portanto, o valor de R$ 69240,00 seria para cobrir os gastos que, caso o projeto não existisse, também não existiriam.

10 CONCLUSÕES

Este trabalho procurou mostrar como o uso de algumas ferramentas e metodologias pode garantir o desenvolvimento de software de melhor qualidade.

Para um software ser de qualidade ele deve satisfazer aos requisitos de seus clientes. Para isso é importante se ter um bom projeto, pois os problemas são muito mais simples e baratos de serem resolvidos ainda na fase de projeto. Um bom projeto provavelmente resultará em um bom software.

Assim como em qualquer outra engenharia também na engenharia de software é importante se medir as coisas, quantificá-las. Através destas medições é possível realizar comparações e estimativas. Estas estimativas são essenciais para o planejamento de projetos, pois permitem saber aproximadamente o tempo que cada atividade irá tomar, quanto irá custar, como deverá ser feito. Às vezes se descobre que o projeto é inviável, ou que demorará muito tempo, ou ainda que não se possuem os recursos técnicos necessários para executá-lo. Portanto quanto mais informações se puderem reunir antes de se começar a codificação, melhor.

A preocupação com a qualidade deve estar presente durante todo o processo de desenvolvimento de um software, assim como de qualquer outro produto. Para garantir a qualidade de qualquer produto este precisa ser testado. Com o software vários testes devem ser executados para garantir um produto que atenda aos requisitos do cliente. Além de apenas testar é importante saber como testar e até quando testar. Softwares muito complexos nunca serão completamente livres de erros, enquanto que softwares que gerenciam sistemas vitais não devem falhar. Cabe ao desenvolvedor determinar a relação entre o bom funcionamento do software e o custo dos testes. O planejamento dos testes contribui para que um maior número de defeitos sejam encontrados no menor tempo.

Terminados os testes o software pode ser entregue ao cliente. O mesmo deve ser acompanhado de completa documentação, explicando tudo que o cliente precisa saber sobre o produto que está adquirindo. O trabalho da empresa, no entanto, ainda não está terminado, uma vez que deverá ser feita a manutenção necessária no software durante toda a sua vida útil e o usuário do mesmo deverá receber o suporte técnico que vier a precisar.

O estudo feito na Theòs Informática Ltda., com a entrega de questionários aos clientes, mostrou como é importante tentar ver o seu produto com os olhos do cliente. Este *feedback* dos usuários com a empresa é fundamental para se desenvolver um produto que atenda às suas necessidades. As reclamações e sugestões obtidas com os questionários foram extremamente importantes para que a empresa delineasse melhor como deveria ser a nova versão de seu software.

O projeto de desenvolvimento do novo SGCP-Paróquia mostrou que não apenas as grandes empresas, mas as pequenas também podem desenvolver um software de qualidade utilizando as ferramentas e metodologias aqui apresentadas. Com elas foi possível estimar os custos, recursos necessários, número de pessoas, tempo de projeto, estabelecer um cronograma a ser seguido, tudo antes de começar a codificação.

É evidente que a complexidade do software e a disponibilidade de recursos por parte da empresa influem no modo de desenvolvimento, o que não significa que se terá um produto ruim. As ferramentas e metodologias aqui apresentadas não trazem custos adicionais às empresas, muito pelo contrário, pois um bom projeto de software permitirá que este seja feito mais rapidamente e que uma vez pronto demande menos manutenção, o que é convertido diretamente em economia para a empresa. Bons profissionais, de preferência com experiência que ajude nas estimativas e que conheçam estas ferramentas e metodologias, são fundamentais.

O maior empecilho que se mostra é a mudança de cultura, uma vez que um planejamento e gerenciamento do desenvolvimento de software exigem disciplina e organização, ambas ausentes em muitas empresas. Às vezes é difícil conter o ímpeto de começar logo a codificação sem elaborar um projeto. Uma vez superada esta barreira o uso dessas e de outras ferramentas e metodologias existentes possibilitam o desenvolvimento de um software confiável, que atenderá aos requisitos do cliente, sendo entregue no prazo estipulado e dentro dos custos projetados, enfim, um software de boa qualidade.

REFERÊNCIAS

CROSBY, P. B. **Qualidade é Investimento**. Rio de Janeiro: José Olympio, 1985.

DEMING, W.E. **Dr. Deming O Americano que Ensinou a Qualidade Total aos Japoneses**. Rio de Janeiro: Record, 1990.

FEIGENBAUM, A.V. ***Total Quality Control***. MacGraw-Hill, 1961.

ISHIKAWA, K. **Controle da Qualidade Total: A maneira Japonesa**. Rio de Janeiro: Campos, 1993.

JURAN, J. M. **Controle da Qualidade**. Handbook. São Paulo: Makron Books, 1991. 1 v.

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de Software**. 1. ed., Editora Novatec, 2006.

KULPA, M. K.; KENT, A. J. ***Interpreting the CMMI: A Process Improvement Approach***. Auerbach Publications, [S.L.], 2003.

PALADINI E. P. **Qualidade Total na Prática. Implantação e avaliação de sistemas de qualidade total**. São Paulo: Atlas, 1994.

PFLEEGER, S. L. **Engenharia de Software: teoria e prática**. 2. ed. São Paulo: Editora Prentice Hall, 2004.

PRESSMAN, R. S. **Engenharia de Software**. 3. ed. São Paulo: Makron Books, 2005.

WEBER, K. C.; ROCHA, A. R. C.; NASCIMENTO, C. J. **Qualidade e produtividade em software**, 4. ed. São Paulo: Makron Books, 2001.

WEBER, K. C. et al. **Modelo de referência e método de avaliação para melhoria de processo de software – versão 1.0**, 4º Simpósio Brasileiro de Qualidade de Software, Porto Alegre (2005).

WEINBERG, G. M. **Software com qualidade: pensando e idealizando sistemas**, 1. ed. Makron Books, 1993.

APÊNDICE A – ANÁLISE DO SGCP-PARÓQUIA

APÊNDICE B – CRONOGRAMA PROPOSTO PARA O PROJETO DO SGCP-PARÓQUIA 5.0

Universidade Estadual de Maringá
Departamento de Informática
Curso de Engenharia de Produção
Av. Colombo 5790, Maringá-PR
CEP 87020-900
Tel: (044) 3261-4324 / 4219 Fax: (044) 3261-5874