UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jair Adelar Brun

Uma Ferramenta Para Automação do Processo de Software Pessoal

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Vitório Bruno Mazzola

Florianópolis, novembro de 2002.

Uma Ferramenta Para Automação do Processo de Software Pessoal

Jair Adelar Brun

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Fernando Álvaro Ostuni Gauthier, Dr.
Coordenador do Curso

Banca Examinadora

Vitório Bruno Mazzola, Dr.
Presidente da Banca

Rosvelter João Coelho da Costa, Dr.

Mário Antonio Ribeiro Dantas, Dr.

Gostaria de agradecer a Vitório Bruno Mazzola,
amigo, companheiro, professor e incansável orientador
deste trabalho. Pela paciência com que me recebeste em Florianópolis,
sempre com boa vontade para auxiliar para que este trabalho tivesse êxito.
A Ivar Luiz Brun e Wanda Brun, meus pais, que sempre estiveram ao
meu lado, dando apoio e incentivo em todos os momentos deste trabalho.
A Sergio Adelar Brun e sua família, pela compreensão e paciência
no período que estive em sua casa, e pelas leituras e sugestões
valiosas na elaboração desta dissertação.
A Graciele Bremm, minha namorada, pela paciência e por
compreender que esta ausência era necessária.
Aos colegas mestrandos, pelas horas boas que passamos juntos.
A todas pessoas que não são mencionadas aqui, mas que incentivaram e
colaboraram para que este trabalho fosse concluído.

LISTA DE ACRÔNICOS

PSP[1] : *Personal Software Process* (Processo de Software Pessoal)
LOC : *Lines of Code* (Linhas de Código)
SEI : *Software Engineering Institute* (Instituto de Engenharia de Software)
KLOC : *One thousand Lines of Code* (mil linhas de código)
PROBE : *Proxy Based Estimating* (Estimativa Baseada em Substitutos)
PIP : *Process Improvement Proposal* (Proposta de melhoria do processo)
KPA : *Key Process Área* (Área chave do processo)
CMM[1] : *Capability Maturity Model* (Modelo de Maturidade da Capacidade)
CASE : *Computer-Aided Software Engineering* (Engenharia de Software Auxiliada por computador)
UML : *Unified Modeling Language* (Linguagem de Modelagem Unificada)
TSP[1] : *Team Software Process* (Processo de Software em Equipe)

[1] *PSP - Personal Software Process, CMM – Capability Maturity Model, TSP – Team Software Process* são marcas registradas da *Carnegie Mellon University*

SUMÁRIO

RESUMO ..... **IX**

ABSTRACT ..... **X**

1 INTRODUÇÃO **11**

1.1 PROBLEMÁTICA 12
1.2 OBJETIVOS 13
1.2.1 Objetivo Geral *13*
1.2.2 Objetivos Específicos *14*
1.3 LIMITAÇÕES DO TEMA 14
1.4 JUSTIFICATIVA 14
1.5 METODOLOGIA 15
1.6 ESTRUTURA DO TRABALHO 16

2 PROCESSO DE SOFTWARE PESSOAL (*PERSONAL SOFTWARE PROCESS*) - PSP **18**

2.1 INTRODUÇÃO 18
2.2 PERSONAL SOFTWARE PROCESS – PSP 20
2.3 O DESENVOLVIMENTO DO PSP 22
2.4 OBJETIVOS DO PSP 24
2.5 OS PRINCÍPIOS DO PSP 25
2.6 POR QUE UTILIZAR O PSP 26
2.7 BENEFÍCIOS DO PSP PARA A ORGANIZAÇÃO 27
2.8 BENEFÍCIOS PESSOAIS DA UTILIZAÇÃO DO PSP 29
2.9 OS FORMULÁRIOS DO PSP 30
2.10 OS NÍVEIS DO PSP 32
2.11 CONCLUSÃO 32

3 OS NÍVEIS DO PSP **34**

3.1 NÍVEL DE MEDIÇÃO PESSOAL 34
3.1.1 Introdução *34*
3.1.2 PSP0 – A Base Do Processo *35*
3.1.3 Os elementos do Processo do PSP *36*
3.1.4 O processo do PSP0 ..... *37*
3.1.5 As medidas do PSP0 ..... *38*
3.1.6 Log de Registro de Tempo *39*
3.1.7 Log de Registro de Defeito *40*
3.1.8 Resumo do Plano de Projeto PSP0 ..... *41*
3.1.9 Conteúdo do PSP0 ..... *41*

3.1.10 Avaliação dos dados do PSP0 42
3.2 PSP0.1 PLANEJANDO O PROCESSO DE SOFTWARE 42
3.2.1 Produzindo um plano de qualidade 42
3.2.2 Os planos de produto são úteis 44
3.2.3 O que é um plano de produto 44
3.2.4 Planejamento de um projeto de software 45
3.2.5 Estrutura de planejamento 46
3.2.6 Medidas de tamanho 47
3.2.7 Conteúdo do processo 49
3.2.8 Novos elementos do processo 50
3.2.9 Avaliação dos dados do PSP0.1 51
3.3 NÍVEL DE ESTIMATIVAS 51
3.3.1 PSP1 - Estimativa De Tamanho 51
3.3.2 Critérios de estimativa de tamanho 52
3.3.3 Estimativa baseada em substituto 53
3.3.4 O método PROBE (PROxy-Based Estimating) 54
3.3.5 Conteúdos do PSP1 60
3.3.6 Novos elementos do processo 61
3.3.7 Avaliação dos dados do PSP1 61
3.4 PSP1.1 PLANEJAMENTO DE TEMPO 62
3.4.1 Estimando o tempo 63
3.4.2 Conteúdo do PSP1.1 64
3.4.3 Novos elementos do processo 64
3.4.4 Avaliação dos dados do PSP1.1 65
3.5 MELHORANDO A QUALIDADE DO SOFTWARE 65
3.5.1 O que são as revisões 66
3.5.2 Revisão de projeto 68
3.5.3 Revisão de código 69
3.5.4 O custo dos defeitos de software 70
3.5.5 Usando a revisão para encontrar defeitos 71
3.5.6 Listas de verificação 72
3.5.7 Conteúdo do PSP2 72
3.5.8 Avaliação dos dados do PSP2 73
3.6 PSP2.1 – PROJETO DE SOFTWARE 74
3.6.1 O projeto conceitual 75
3.6.2 O Modelo de Especificação de Estado 76
3.6.3 O Modelo de Especificação Lógica 77
3.6.4 O Modelo de Especificação funcional 77
3.6.5 O Modelo de Cenário Operacional 78

3.6.6 Conteúdo do PSP2.1 ... *78*
3.6.7 Avaliação dos dados do PSP2.1 ... *79*
3.7 O PROCESSO CÍCLICO ... 80
3.7.1 O Desenvolvimento Cíclico ... *80*
3.7.2 Planejamento e Requisitos ... *81*
3.7.3 Projeto de Alto Nível ... *82*
3.7.4 Desenvolvimento Cíclico ... *82*
3.7.5 Teste de Integração do Sistema ... *83*
3.7.6 Reavaliação e novo ciclo ... *83*
3.7.7 Conteúdo do PSP3 ... *84*
3.7.8 Novos elementos do processo PSP3 ... *85*
3.7.9 Relatando especificações do processo ... *85*
3.8 CONCLUSÃO ... 86
4 UMA FERRAMENTA DE SUPORTE À APLICAÇÃO DO PSP ... 88
4.1 INTRODUÇÃO ... 88
4.2 ANÁLISE DE REQUISITOS ... 88
4.2.1 Descrição do produto ... *88*
4.2.2 Objetivo do sistema ... *89*
4.2.3 Objetivos Específicos do sistema ... *89*
4.2.4 Informações do sistema ... *89*
4.3 REQUISITOS DO SISTEMA ... 91
4.4 DETALHAMENTO DO MODELO ... 91
4.4.1 Especificação do Modelo ... *93*
4.4.2 Implantação do sistema ... *109*
4.4.3 Descrição das Classes e atributos ... *115*
5 CONCLUSÕES E PERSPECTIVAS PARA TRABALHOS FUTUROS ... 117
5.1 CONCLUSÃO ... 117
5.2 PERSPECTIVAS DE TRABALHOS FUTUROS ... 118
5.3 BIBLIOGRAFIA ... 120

LISTA DE FIGURAS

Figura 2.1 O CMM e o PSP 23
Figura 3.1 Fluxograma do Processo do PSP0 35
Figura 3.2 O Processo do PSP 37
Figura 3.3 Estrutura de planejamento de Processo 46
Figura 3.4 O Processo do PSP3 81
Figura 4.1 Diagrama de Caso de Uso "Registrar Projeto PSP" 94
Figura 4.2 Diagrama de Classe Registrar Projeto PSP 98
Figura 4.3 Diagrama de Seqüência do caso de uso Registrar Projeto PSP 98
Figura 4.4 Diagrama de Caso de Uso "Registrar Defeitos Corrigidos" 99
Figura 4.5 Diagrama de Classe Defeitos Corrigidos 102
Figura 4.6 Diagrama de Seqüência Defeitos Corrigidos 103
Figura 4.7 Diagrama do caso de uso Registrar Tempos 104
Figura 4.8 Diagrama de Classes Registro de Tempos 105
Figura 4.9 Diagrama de Seqüência Registro de Tempos 106
Figura 4.10 Diagrama de caso de Registrar Resumo Projeto 107
Figura 4.11 Diagrama de Classe Resumo Projeto 108
Figura 4.12 Diagrama de seqüência Resumo Projeto 109
Figura 4.13 Login no Sistema 109
Figura 4.14 Registrar Sistema 110
Figura 4.15 Registrar Engenheiro 110
Figura 4.16 Registrar Nível PSP 111
Figura 4.17 Registrar Tipos de Defeitos 111
Figura 4.18 Registrar Fases do Processo 112
Figura 4.19 Registrar Projeto PSP 112
Figura 4.20 Registrar Tempos 113
Figura 4.21 Resumo PSP0 - Tempo nas Fases 113
Figura 4.22 Resumo PSP0 - Defeitos Injetados 114
Figura 4.23 Resumo PSP0 - Defeitos Removidos 114

RESUMO

O *Personal Software Process* (Processo de Software Pessoal – PSP) é um conjunto de *scripts*, formulários, modelos e padrões desenvolvidos para prover disciplina ao processo de desenvolvimento de software. Este trabalho faz uma análise dos procedimentos envolvidos na utilização do PSP como estimativas de tempo e tamanho, planejamento de produtos, revisão de projeto e código e do processo de desenvolvimento cíclico no qual grandes programas são divididos em programas menores que variam entre 100 e 300 linhas de código. Devido às mudanças que ocorrem no processo pessoal de desenvolvimento de software, o PSP é dividido em sete níveis, sendo apresentado gradualmente ao engenheiro de software, para diminuir o impacto das mudanças necessárias no processo de desenvolvimento. Devido ao grande número de formulários e cálculos envolvidos na utilização do PSP, este trabalho apresenta uma proposta de ferramenta para suporte automatizado à utilização do PSP. Esta ferramenta é utilizada para eliminar a cópia e transferência de dados entre formulários, evitando erros durante a transcrição. Outro benefício é a realização automática dos cálculos requeridos, facilitando a correção de algumas informações incorretas que possam ter sido registradas. Esta ferramenta também permite que as informações dos formulários sejam recuperadas de maneira rápida e precisa. A duplicação dos dados nos diversos formulários é eliminada com a utilização desta ferramenta, o que os torna mais consistentes.

PALAVRAS-CHAVE: Processo de Software Pessoal, Engenharia de Software, Qualidade de Processo.

ABSTRACT

The Personal Software Process - PSP, is a set of scripts, forms, models and standards, developed to provide discipline to the software development process. The objective of this paper is to analyze the procedures involved in the usage of PSP, like estimates of time and length, product planning, the review of the project and code, and the cyclic development process, where large programs may be divided into smaller ones with about 100 to 300 code lines. Because of the changes that happen by the personal development of software, PSP is divided in seven levels that may be gradually presented to the software engineer, to cause less impact by the changes that are necessary during the process of development. Because of the great amount of forms and calculus involved in the usage of PSP, this paper presents a proposal of a tool that gives automatic support to the use of PSP. This tool may be used do eliminate copies and transferences from one form to another, avoiding mistakes during the transcription. Another benefit is that the programs accomplish automatically the calculus that is required, making it easy to correct mistakes. This tool also permits to get information back from the forms quickly and directly, and double information in different forms may be eliminated, making them more consistent.

KEYWORDS: Personal Software Process, Software Engineering, Quality of Process.

1 INTRODUÇÃO

Atualmente, grande número de pessoas depende de aplicações de software para a realização das atividades cotidianas. O software faz parte da nossa vida e, apesar da evolução dos últimos 30 anos, ainda há muito por fazer. Principalmente na busca de qualidade e produtividade no desenvolvimento e manutenção de software. Atualmente, o principal objetivo da engenharia de software é melhorar a qualidade do software (ROCHA *et al*, 2001).

Na incessante busca por melhorias na qualidade de software, identifica-se que a qualidade do processo de desenvolvimento é tão importante quanto a qualidade do produto desenvolvido. Assim, principalmente durante a década de 90 houve grande preocupação com a modelagem e melhorias no processo de software (ROCHA et al., 2001). Visando a evolução da qualidade do processo de desenvolvimento, Watts Humphrey desenvolveu o *Personal Software Process* (Processo de Software Pessoal - PSP). O PSP tem por objetivo auxiliar os engenheiros a melhorar o processo de desenvolvimento através da melhora de seu processo pessoal.

O *Personal Software Process* (Processo de Software Pessoal – PSP) é um conjunto estruturado de formulários, relatórios e *scripts* que visam dar disciplina ao engenheiro de software. Disciplina é definida como uma atividade ou exercício que desenvolve ou melhora as habilidades. Ao contrário do que se imagina, a disciplina na engenharia de software não existe apenas para onerar o trabalho, existe para entender e melhorar o trabalho pessoal. A disciplina no PSP provê uma estrutura para desenvolver as habilidades pessoais e, métodos para auxiliar a ser um engenheiro de software. A disciplina no PSP acelera a sua aprendizagem (HUMPHREY, 1997).

A disciplina está na utilização de *scripts*, formulários e modelos. Eles proporcionam a estruturação e definição do processo de desenvolvimento de software. Este processo estruturado e definido para desenvolvimento de software é a seqüência dos passos necessários para desenvolvimento e manutenção de um software (HUMPHREY, 1995).

Os formulários auxiliam o engenheiro a conhecer seu trabalho através de medidas pessoais. Podem ser averiguados medidas de tempo, defeitos, rendimento das revisões de projeto, código e outros. A medida de tempo é realizada para saber onde e como é

gasto o tempo de desenvolvimento. As medidas de defeitos são utilizadas para acompanhar os defeitos injetados e removidos durante o desenvolvimento de um produto de software.

Existem outros formulários, *scripts*, padrões e relatórios, em sete níveis diferentes. Os formulários de medidas de tempo e defeitos situam-se no primeiro nível e, também, são utilizados em todos os outros níveis. Os demais controles são inseridos na medida em que os engenheiros evoluem nos níveis do PSP. É proposta deste trabalho oferecer uma ferramenta de suporte automatizado para ser usada por engenheiros de software que fazem uso do PSP.

Esta proposta deriva da premissa de que o planejamento, controle de defeitos, controle de tempos, estimativas de tamanhos, recursos e revisões durante o desenvolvimento de software, proporcionam maior compreensão e domínio do processo de desenvolvimento. Isto conduz a elevação substancial do rendimento do processo de desenvolvimento de software. O PSP pode ser trabalhado manualmente ou, também, de forma automatizada. A automação do PSP é a proposta efetiva. Elaborar-se-á uma proposta de ferramenta para suporte automatizado dos processos que o envolvem, além da teorização inerente e necessária.

Faz-se o levantamento do estado da arte, surgimento, objetivos, benefícios para a organização e para o engenheiro de software. Também será analisado como e o que o PSP envolve em termos de estimativas, planejamentos, revisões de projeto e código, medidas pessoais e processo definido.

1.1 PROBLEMÁTICA

A proposta de uma ferramenta de suporte automatizado do PSP surgiu após análise e constatação de que existem mais de dez formulários que precisam ser preenchidos e analisados, para que os objetivos do PSP sejam alcançados. Além disso, há a preocupação com o tempo consumido, a quantia de papel e cálculos manuais envolvidos na utilização do PSP.

Um sistema de software desenvolvido usando PSP2.0, por exemplo, requer que sejam preenchidos mais de dez formulários de papel, incluindo um resumo de plano de projeto, log de registro de tempo e defeito, proposta de melhoria de processo, modelos

de estimativa de tamanho, estimativa de tempo, relatório de teste, planejamento de tarefa, planejamento de horário, lista de verificação de projeto e lista de conferência de código.

Estes formulários, caracteristicamente, rendem um grande número de valores distintos e, cada engenheiro calcula e registra manualmente para um único projeto. Este é um trabalho tedioso e propenso a erros, porém o trabalho do gerente na conferência e análise de todos estes valores pode ser muito mais complexo.

O trabalho de preenchimento e análise de formulários do PSP pode ser feito de forma manual, como têm sido feito até hoje. Tem-se o exemplo de um instrutor em um curso de PSP, que fez a conferência manual de mais de 31.000 dados do PSP, avaliados para os nove projetos desenvolvidos por 10 estudantes, além de conferir o projeto de software deles (JOHNSON & DISNEY, 1998).

Não há nada errado em fazer a análise e conferência dos dados de forma manual, quando os fins justifiquem os meios. No caso do PSP manual, porém, é realizado um trabalho que envolve grande esforço do engenheiro. É preciso saber se, quando feito à mão, a melhoria proposta pelo PSP pode ser afetada, no direcionamento do engenheiro a uma significante melhora de processo (JOHNSON & DISNEY, 1998).

A automação do PSP pode aumentar substancialmente o rendimento de todo o processo. Permite, também, ao engenheiro utilizar o tempo usado nos cálculos e análise dos dados em outras atividades, que podem ser mais produtivas. Além disso, o processo manual é muito mais propenso a erros, trabalhoso, oneroso e consumidor de tempo.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo geral desta dissertação é a proposição de uma ferramenta para suporte automatizado dos cálculos e análises de dados gerados pela utilização do PSP. Esta ferramenta deve auxiliar, o engenheiro de software, na produção de dados para análise do processo de desenvolvimento de maneira mais rápida e segura.

O objetivo da ferramenta é a automação do processo do PSP. Deve automatizar os cálculos e gerar as informações que o engenheiro de software necessita para avaliar e melhorar o seu trabalho.

1.2.2 Objetivos Específicos

1.2.2.1 Revisão bibliográfica sobre o *Personal Software Process* (Processo de Software Pessoal – PSP) e os assuntos relacionados a este.

1.2.2.2 Propor uma ferramenta para suporte automatizado do PSP que auxilie a análise dos dados dos engenheiros de software. Permitindo, a estes, a obtenção das informações sobre sua performance de maneira automática. Facilitando a análise dos dados providos pela utilização do PSP.

1.3 LIMITAÇÕES DO TEMA

O trabalho visa a elaboração de uma proposta de ferramenta de suporte automatizado para utilização do *Personal Software Process*. Não é intenção deste trabalho analisar e executar os seguintes itens:

a) *Team Software Process* – TSP - Processo de Software em Equipe.

b) Implementação da proposta em uma linguagem de programação.

c) Testes com usuários para validação da ferramenta.

1.4 JUSTIFICATIVA

A automação do PSP pode tornar seus cálculos e análises mais efetivas. O registro de tempos pode ser feito de forma automática, permitindo que o cálculo de tempo gasto em cada tarefa seja realizado automaticamente. A automação pode realizar os cálculos muito mais rapidamente, do que se fosse feito à mão. Além disso, pode ser feito com menor número de erros.

A análise dos dados providos pelos PSP, quando realizado com papel e caneta, manual, leva ao *stress* pessoal, induz e é mais propenso a erros. Quando ocorre erro nos cálculos em um dos formulários, o número de cálculos que precisam ser refeitos pode ser muito grande, muitas vezes todos os cálculos seguintes, até o final.

A automação facilita a avaliação dos dados providos pelos formulários. A redundância de informações pode ser reduzida ou até mesmo eliminada. Os dados são armazenados num só lugar, de maneira que possam ser recuperados, de acordo com as necessidades. Elimina a cópia e transferência de dados, manual, entre formulários, que é propício ao erro. Esta ferramenta é desenvolvida em um ambiente multitarefa, o que permite, ao usuário, alternar entre a ferramenta e o ambiente de desenvolvimento.

Ferramentas para suporte automatizado podem ajudar bastante, mas, sós, não resolvem os problemas da engenharia de software. Sequer o processo sozinho é capaz de resolver estes problemas. Ambos são necessários para alcançar os objetivos de melhoramento da qualidade. Ferramentas melhor elaboradas são necessárias em muitos aspectos do processo de software. Elas podem aumentar a produtividade, reduzir erros, simplificar tarefas rotineiras e liberar os engenheiros para tarefas mais criativas.

Ferramentas de apoio podem tornar o PSP mais e eficiente e fáceis de utilização. Ambientes *CASE* (*Computer Aided Software Engineering*). Ou, traduzindo: Engenharia de Software Auxiliada por Computador, são necessários para auxiliar os métodos do PSP no local de trabalho dos engenheiros, em adição a todas as outras ferramentas já utilizadas. Ferramentas, como esta, podem facilitar o registro de tempo, acompanhar defeitos, manutenção dos dados e cálculos estatísticos que tornam os métodos do PSP mais fáceis de serem aprendidas e utilizadas.

1.5 METODOLOGIA

O trabalho que se propõe é a construção de um projeto do sistema para automação da execução do PSP. A finalidade é o apoio às atividades de analisar e melhorar o desenvolvimento de sistemas. Definiram-se os componentes necessários para o projeto utilizando *Unified Modeling Language* (Linguagem de Modelagem Unificada – UML).

Para a elaboração do projeto da ferramenta será realizada uma pesquisa bibliográfica que "...é aquela que dá ênfase à consulta a livros para a obtenção de dados

necessários para a devida explicação e compreensão do tema em foco " (NÉRICI, 1993). Estrutura-se um conjunto de diagramas (*use case*, diagrama de classe e diagrama de seqüência), definindo a proposta da ferramenta de suporte automatizado do PSP.

A bibliografia é particularmente útil para a apreensão dos conceitos básicos do Processo de Software Pessoal – PSP: medidas pessoais, planejamento, estimativas de tamanho e tempo, revisão de código, projeto e outros. É útil, também, para a estrutura a ser utilizada pelo PSP que são: formulários, padrões, modelos e *scripts*. A observação do funcionamento do PSP também se constitui em valioso instrumento para a compreensão do problema.

1.6 ESTRUTURA DO TRABALHO

Para a consecução dos objetivos propostos, o presente trabalho está estruturado em 8 capítulos, incluindo a introdução. O presente capítulo faz a introdução ao trabalho, delineia os objetivos e limitações e as justificativas. Apresenta, também, a metodologia de desenvolvimento do estudo e a estrutura do trabalho.

No capítulo 2 é feita a introdução ao *Personal Software Process* (Processo de Software Pessoal) – PSP. Expõe-se sobre o surgimento, estratégias utilizadas para o desenvolvimento e os benefícios alcançados pelas organizações de software e engenheiros de software.

No capítulo 3 são apresentados os níveis do PSP. O nível de medição pessoal (PSP0 e PSP0.1), é onde são registradas as medidas de tempos utilizados em cada uma das fases do ciclo do desenvolvimento e, quais defeitos foram encontrados e corrigidos. Isto só é possível com a utilização de formulários adequados. Estas medidas servem de referência para avaliação do processo e proposição de melhorias. No PSP0.1 é apresentado o formulário de proposta de melhoria do processo, que é utilizado para fazer o registro das idéias e problemas que podem ocorrer durante o processo de desenvolvimento. O nível de estimativa pessoal (PSP1 e PSP1.1), o engenheiro de software vê a forma de se fazer estimativas de tamanho do programa e tempo de desenvolvimento. A idéia geral é saber estimar quanto tempo se gasta para realizar uma tarefa, com base em medições feitas em tarefas semelhantes, feitas anteriormente. Neste nível, consegue-se analisar os compromissos e assumir somente aqueles que possam

realmente ser cumpridos. O nível PSP1.1 inclui o planejamento de tarefas e a elaboração de cronogramas. No nível de qualidade pessoal (PSP2 e PSP2.1) é feito o estudo sobre revisão de projeto e código. As revisões são utilizadas para conseguir identificar e remover os erros nas fases iniciais do processo de desenvolvimento. Para a realização das revisões, o engenheiro deve fazer uso de uma lista de verificação (*checklist*), com os itens a serem revisados. O nível PSP2.1 inclui a elaboração de modelos e padrões de projeto, bem como métodos de análise e acompanhamento do projeto. Seqüencialmente, é apresentado o nível de processo cíclico de desenvolvimento (PSP3) que é a última etapa do PSP. Neste nível, o PSP afasta-se do desenvolvimento de pequenos programas para tratar do desenvolvimento de projetos maiores, embora, ainda, em nível pessoal. A estratégia do PSP3 é subdividir um programa maior em programas menores de tamanho requerido pelo PSP2.1. Divide-se, assim, em programas que variam de 100 a 300 linhas de código (*Lines of Code* – LOCs). Para cada programa menor, faz-se um PSP2.1 completo. Inclui-se, neste, o projeto, a codificação, a revisão, a compilação e os testes. Assim, o PSP3 consegue atender programas de até várias mil LOC (KLOC).

No capítulo 5 é apresentada a proposta da ferramenta para suporte automatizado do PSP. São desenvolvidos alguns casos de uso, diagramas de classe e diagrama de seqüência dos principais itens do PSP.

No capítulo 6 são apresentadas as conclusões do trabalho, considerações, bem como sugestões para trabalhos futuros, diagnosticadas a partir da análise crítica de fatores diversos e ferramentas pesquisadas.

Os Quadros contendo os *scripts*, modelos e formulários, referenciados durante o texto, necessários em todos os níveis do PSP, são disponibilizados apenas por meio eletrônico. Cópia destes ficam disponíveis junto à Secretaria de Pós-Graduação da Universidade Federal de Santa Catarina e na Biblioteca Central desta mesma instituição. Podendo ser consultado junto à Biblioteca Central.

2 PROCESSO DE SOFTWARE PESSOAL (*PERSONAL SOFTWARE PROCESS*) - PSP

2.1 INTRODUÇÃO

O desenvolvimento de software normalmente se dá de forma não estruturada. O sucesso do sistema depende da capacidade e da criatividade individual do programador. Em todo o processo deve se fazer presente a qualidade durante todo o projeto. A conformidade a requisitos de funcionalidade e desempenho explícitos e implícitos é definida como qualidade de software (PRESSMANN, 1995).

A falta de planejamento e de processos estruturados, pode acarretar atraso na entrega do produto e, conseqüente aumento do custo do software desenvolvido. Isto torna a qualidade do produto a ser entregue, imprevisível. Geralmente, aspectos de qualidade são tratados quando o produto já está pronto, ou seja, tenta-se implantar qualidade em um produto desenvolvido em um processo de qualidade duvidosa.

Este modo de desenvolvimento de software em determinadas situações pode ser desastroso e, as correções no produto normalmente têm seu custo mais alto que o previsto no projeto. Estes métodos não estruturados de desenvolvimento de software, só são aceitos na não existência de alternativas. O método de desenvolvimento de software sempre esteve mais próximo de uma atividade artesanal do que de uma disciplina de engenharia de software.

Os engenheiros de software, normalmente, desenvolvem métodos e técnicas próprias para o desenvolvimento de software. Alguns dos produtos de software já concluídos podem até funcionar, mas geralmente isto ocorre após extensa e demorada atividade de testes e reparos. Esta situação fica crítica quando a contribuição de cada indivíduo é exclusivamente importante.

Esta idéia pode ser melhor representada por um coral de vozes. O desempenho geral do coral é um conjunto harmonioso de vozes e, cada músico deve ser um contribuinte altamente competente e disciplinado. Artistas individuais se sobressaem ocasionalmente, mas, o coral é muito mais que a soma dessas partes e, uma única nota errada de qualquer um dos integrantes pode prejudicar o desempenho de todo o grupo.

Esta situação pode ocorrer no desenvolvimento de software. O desempenho global dos indivíduos é extremamente importante. Porém, se apenas um engenheiro de software cometer erros em algum dos procedimentos, isto pode afetar o desempenho de toda equipe de desenvolvimento e do projeto como um todo.

Isto é um grande problema para as organizações empresariais e empresas de desenvolvimento de software. Para tanto, busca-se a resolução destes problemas, com a utilização do *Personal Software Process* (Processo de Software Pessoal) – PSP. O PSP é um conjunto de formulários, procedimentos e relatórios que auxiliam o processo individual do engenheiro de software, com base em conceitos clássicos de qualidade.

Entre os conceitos mais importantes de qualidade encontra-se o da garantia de qualidade, oposto ao controle de qualidade. Enquanto o controle de qualidade busca encontrar defeitos nos produtos acabados, a garantia de qualidade procura garantir que, em cada etapa do desenvolvimento do produto, os defeitos não sejam injetados (COSTA & SANTANA FILHO, 2000).

Uma organização disciplinada de engenharia de software possui práticas definidas, seus profissionais usam essas práticas, são monitorados e se esforçam para melhorar o desempenho, sentem a responsabilidade pela garantia da qualidade. Isto faz com que eles tenham confiança e dados necessários para resistir a demandas irracionais de compromissos.

Para melhorar a qualidade dos softwares desenvolvidos há a necessidade de capacitar os engenheiros de software. Faz-se com que eles conheçam sua própria capacidade e, em conseqüência disto, consigam assumir compromissos que possam ser cumpridos. Dentre estes, destaca-se cronogramas dentro de um prazo que possa ser atendido e custos que não ultrapassem o orçamento.

Histórias de projetos de software fracassados que aparecem na literatura poderiam ter tido o final diferente. Para isto, dentre outras providências, precisariam ter sido efetivadas as medições do processo de desenvolvimento, como número de defeitos injetados, tamanho do produto desenvolvido etc. Estas medições deveriam ser vistas como algo que traz resultados efetivos ao processo, e não consideradas primordialmente como uma atividade consumidora de tempo e recursos (PRESSMANN 1995).

2.2 *PERSONAL SOFTWARE PROCESS – PSP*

O *Personal Software Process* (Processo de Software Pessoal) – PSP, é um processo de auto-melhoria desenvolvido para auxiliar o engenheiro de software a controlar, administrar e melhorar seu modo de trabalhar. O PSP possui uma estrutura de formulários, diretrizes e procedimentos que auxiliam no desenvolvimento de software. Através do PSP o engenheiro pode desenvolver um processo estruturado para o desenvolvimento de software (HUMPHREY, 1995).

Quando utilizado corretamente, o PSP provê dados históricos que o engenheiro de software utiliza para planejar e assumir seus compromissos. O PSP pode tornar elementos rotineiros do trabalho, do engenheiro de software, mais previsíveis e eficientes. Com a utilização do PSP o engenheiro de software pode descobrir onde estão os problemas de desenvolvimento e como corrigí-los (HUMPHREY, 1995).

Para que o engenheiro de software consiga melhorar o seu trabalho, ele precisa conhecer a sua potencialidade. Este conhecimento é proporcionado pelo PSP através de medidas pessoais do trabalho do engenheiro. Com a utilização de medidas do processo pessoal, como tempo de trabalho e erros injetados e removidos do software, o engenheiro consegue fazer uma análise do seu trabalho. Nesta, pode ser verificada onde é gasto o tempo de desenvolvimento, onde são injetados e removidos os erros do sistema etc.

Entretanto, o PSP não é uma solução mágica aos problemas de desenvolvimento de software. Embora, possa dar subsídios e mostrar onde e como melhorar, o engenheiro tem que fazer as melhorias por si próprio. O propósito exclusivo do PSP é ajudar o engenheiro de software a fazer um trabalho melhor (HUMPHREY, 1995).

O PSP é uma ferramenta poderosa que pode ser utilizada com vários objetivos. Por exemplo: administrar o trabalho, avaliar o próprio talento e construir habilidades de desenvolvimento de software. Além disso, pode auxiliar no planejamento das tarefas, descobrir precisamente o próprio desempenho e aumentar a qualidade dos produtos.

Em vez de usar um modelo de desenvolvimento para cada trabalho, os engenheiros de software devem utilizar um conjunto sofisticado de ferramentas, métodos e conhecimento. O PSP proporciona dados e técnicas de análise que podem ser utilizadas para determinar que tecnologia adotar e qual o melhor método de trabalho.

O PSP também possui um formulário para registro dos erros cometidos. Este, posteriormente pode ser utilizado para descobrir os erros que foram cometidos, onde estes erros foram injetados e removidos, e como corrigí-los. Esta análise dos erros de desenvolvimento auxilia para que os mesmos erros não sejam cometidos novamente.

Uma outra atividade do PSP é o processo de revisão de projeto e código. Com a utilização destas revisões é possível aumentar a qualidade do produto desenvolvido. A qualidade destas revisões é determinada pelo próprio engenheiro de software. Nela, ele verifica os tipos de erros que não são encontrados e o método de revisão que se mostra mais efetivo.

Existe uma preocupação por parte dos engenheiros em relação ao uso de um processo de desenvolvimento disciplinado, alegam que isto pode restringir sua criatividade. Eles imaginam que o planejamento das tarefas, por exemplo, limita a criatividade no desenvolvimento de idéias novas e originais. As observações sugerem exatamente o contrário (HUMPHREY, 1995).

Parte do tempo de desenvolvimento é gasto na resolução de problemas que já ocorreram diversas vezes, porém, por falta de dados históricos, estes problemas voltam a ser repetidos em projetos subseqüentes. As organizações de software gastam tempo resolvendo problemas similares aos resolvidos nos projetos anteriores. Um número infinito de horas é gasto na depuração e resolução destes problemas, aí vem a pergunta: Isto é criatividade? A complexidade de um problema não significa que a solução seja criativa. Criatividade exige originalidade e idéias diferentes (CRISTIANO et al., 2002).

Após os engenheiros de software gastarem muitos anos, resolvendo os mesmos problemas, antigos e conhecidos, alguns sentiram que seu tempo foi perdido. Suas energias criativas podem ser melhor aplicadas na busca e solução da causa do problema, ao invés de reparar as conseqüências de maneira repetitiva. A experiência sugere que o engenheiro de software pode fazer isto, efetivamente, quando tiver um processo medido e definido (HUMPHREY, 1995).

O planejamento é parte importante na execução de um projeto com sucesso. Na elaboração de um plano de projeto, é desenvolvido o planejamento para a execução do trabalho e estimado quanto tempo se gasta para a sua conclusão. Neste, deve-se analisar as tarefas a serem desenvolvidas, como estas são executadas, qual tarefa se executa

primeiro, e as subseqüentes, bem como, quanto tempo se gasta na execução do que foi planejado.

Os engenheiros de software precisam aprender a projetar seus programas antes de iniciar a codificação. Com a análise do projeto antes da implementação, podem ser economizados tempo e evitados muitos erros. Um pouco de visão ordenada do futuro, pode economizar muito tempo (HUMPHREY, 1997).

Outro benefício importante do planejamento pessoal, diz respeito ao prazo de entrega. Há engenheiros de software que assumem prazos de entrega de um produto, difíceis de serem cumpridos. Isto ocorre em parte das organizações de software. O problema é que, geralmente, os clientes somente identificam o que eles querem, quando eles precisam disto. Poucos conseguem antecipar suas necessidades com bastante antecedência para permitir um prazo de desenvolvimento realista.

Com um plano detalhado e preciso, é possível entender quanto tempo de trabalho é necessário para completar uma tarefa. Com o suporte de dados demonstra-se que o plano é sólido, realista e leva em conta sua experiência. Pode ser possível demonstrar o que é necessário para acelerar o desenvolvimento e o que pode causar atraso no projeto.

Gerentes e clientes querem o engenheiro de software trabalhando agressivamente e, terminando o produto com a maior brevidade possível. Também querem prazos com os quais possam contar. Com um plano sólido, é possível dar-lhes convicção. E, quando existe convicção, é possível sustentar a opinião nestes debates de compromisso (HUMPHREY, 1995).

2.3 O DESENVOLVIMENTO DO PSP

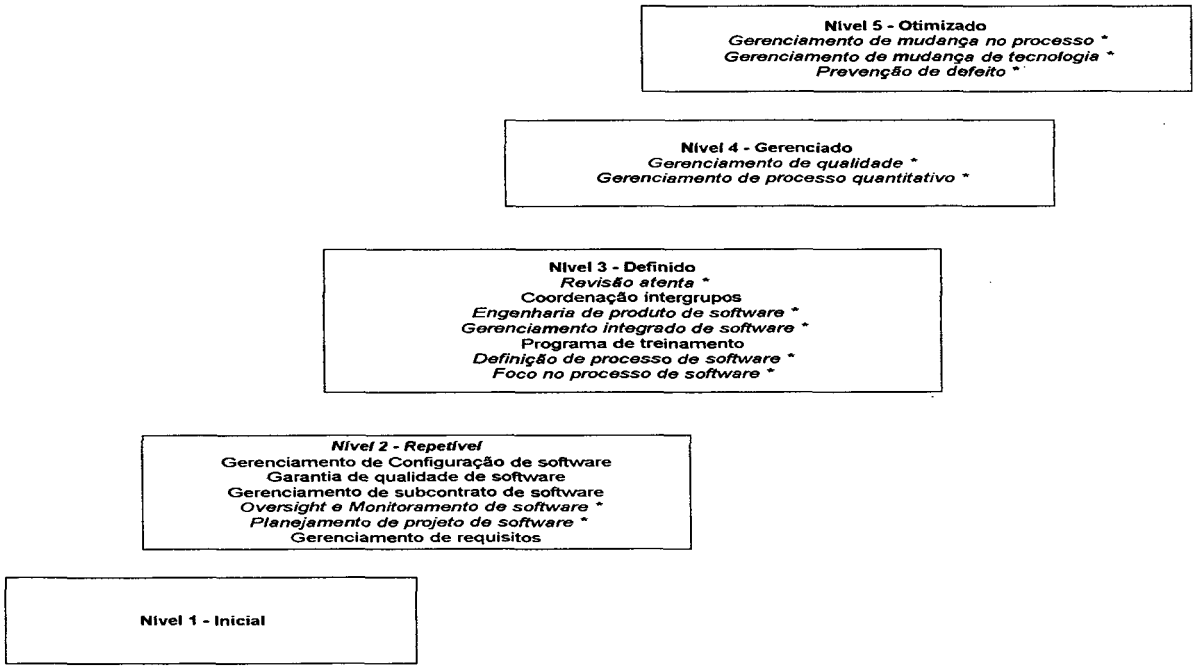
O Modelo de Maturidade da Capacidade – CMM, foi desenvolvido pelo *Software Engineering Institute* (SEI), ligado à Universidade *Carnegie Mellon*, com o objetivo de estabelecer um padrão de qualidade de software para as forças armadas. No CMM são estabelecidos cinco níveis de maturidade referentes à capacidade que uma organização ou empresa de software possui para desenvolver software. Estes níveis são: inicial, repetível, definido, gerenciado e otimizado. Cada um desses níveis possui algumas áreas chaves do processo, com exceção para o nível inicial que não possui nenhuma área chave (PAULK et al., 1993).

O CMM mostra às grandes organizações de desenvolvimento de software o que deve ser feito e não, como fazer. Algumas pessoas questionaram Humphrey sobre como aplicar o CMM nas pequenas organizações ou para pequenas equipes de software. Enquanto os princípios de CMM são aplicados para grandes organizações, estava faltando um modelo para pequenas organizações e engenheiros de software (HUMPHREY, 1995).

Humphrey, então decidiu usar princípios do CMM pessoalmente, para desenvolver um modelo de tamanho do programa, ver como a proposta se comporta e para entender como convencer o engenheiro de software para adotar tal prática. Desenvolvendo programas pequenos, usou todas as práticas do CMM, do Nível 1 até o Nível 5.

A Figura 2.1 mostra o CMM e as áreas chaves do processo que são pelo menos parcialmente resolvidas pelo PSP, destacadas em itálico e com asterisco. Algumas áreas chaves do processo do CMM são excluídas por não se adequarem ao nível individual.

Figura 2.1 O CMM e o PSP



Fonte: (HUMPHREY, 1995) p. 7

Durante três anos, foi desenvolvido um total de 62 programas e definidas aproximadamente 15 versões do modelo de PSP. Foram utilizadas as linguagens de programação Pascal, Object Pascal e C++ para desenvolver aproximadamente 25.000 linhas de código. Desta experiência, conclui-se que os princípios de processo da administração da qualidade de Deming (1990) e Juran (1992), são da mesma maneira, aplicáveis ao trabalho do engenheiro de software individual, bem como a outros campos de tecnologia.

No desenvolvimento do PSP foi utilizada a seguinte estratégia HUMPHREY, 1995):

- identificar práticas e métodos de grandes sistemas de software que pudessem ser utilizados pelos indivíduos;
- definir um subconjunto dessas práticas e métodos que podem ser aplicados durante o desenvolvimento de pequenos programas;
- estruturar essas práticas e métodos para serem introduzidos gradualmente;
- prover exercícios para praticar esses métodos em um ambiente de aprendizagem.

2.4 OBJETIVOS DO PSP

O PSP foi desenvolvido pelo *Software Engineering Institute* (Instituto de Engenharia de Software) – SEI com os seguintes objetivos (HUMPHREY, 1995):

- prover um conjunto de métodos de desenvolvimento que possam ser repetidos em projetos subseqüentes;
- auxiliar o engenheiro de software a medir, planejar, registrar e analisar seu trabalho;
- prover dados históricos que auxiliam a fazer estimativas, possibilitando assumir e cumprir os compromissos;
- permitir ao engenheiro de software, aprender a partir da variação de sua performance, fazendo com que se torne apto a melhorar seu trabalho;
- estabelecer uma metodologia para melhorar, a nível pessoal, a capacidade de planejamento, acompanhamento e qualidade dos trabalhos;

- o PSP pode ser utilizado como ferramentas de uso geral para gerenciar as atividades pessoais particulares e profissionais;
- criar um comprometimento pessoal para a qualidade;
- envolvimento contínuo do engenheiro na melhoria do processo.

Os objetivos propostos pelo PSP, são para a melhora do trabalho individual e organizacional. Provendo métodos e organização para que o engenheiro de software possa fazer um trabalho bom no desenvolvimento de software. Além de auxiliar no processo profissional, o PSP pode ser utilizado em situações particulares. Como por exemplo, na elaboração de uma dissertação de mestrado. Faz-se o planejamento, as estimativas, o projeto o desenvolvimento e a conclusão do trabalho. Medidas de tempo e defeitos podem ser facilmente aplicadas a esta situação.

2.5 OS PRINCÍPIOS DO PSP

O desenvolvimento do PSP tem por base o seguinte planejamento e princípios de qualidade (HUMPHREY, 2000):

- cada pessoa é diferente, por isso, para ser mais efetivo precisa planejar o seu trabalho, baseando seus planos em dados pessoais;
- para melhorar o seu desempenho constantemente, os engenheiros devem usar processos bem definidos e medidos;
- para desenvolver produtos de software de qualidade, os engenheiros precisam sentir-se responsáveis na qualidade dos produtos;
- as pessoas só assumem compromissos pessoais voluntariamente – imposições não são compromissos; podem ser até aceitas, porém como obrigações e não como compromissos;
- custa menos para achar e corrigir erros nas fases iniciais do desenvolvimento, do que em processos posteriores;
- é mais eficiente prevenir defeitos do que procurá-los e corrigí-los;
- o jeito certo sempre é o modo mais rápido e mais barato para fazer um trabalho;
- cronogramas e planos corporativos podem não ser vistos pelos engenheiros de software como compromisso pessoal.

Para que um engenheiro de software trabalhe corretamente, há a necessidade de planejar seu trabalho antes de iniciá-lo. Além disso, é preciso fazer uso de um processo de desenvolvimento definido. O engenheiro de software pode entender o seu desempenho pessoal, com a utilização de medições do tempo gasto em cada etapa do processo, defeitos injetados e removidos, bem como o tamanho dos produtos produzidos.

Para produzir constantemente produtos de qualidade o engenheiro de software precisa conhecer a sua performance. Para que possa fazer isto, é necessário planejar, medir e registrar a qualidade do produto. O foco em qualidade deve ser mantido desde o princípio do trabalho. Por fim, é preciso completar o trabalho com a análise dos dados produzidos, fazendo uso do resultado desta análise para melhorar os processos pessoais, continuamente.

2.6 POR QUE UTILIZAR O PSP

Alguns projetos de desenvolvimento de software são infestados com problemas de qualidade, custo e cronograma estourado. Alguns destes problemas são de nível individual. Por exemplo, problema de defeito na qualidade de software que foram entregues, podem resultar de falhas de projetos de baixa qualidade e lógica algorítmica feitos por indivíduos, assim como a demora na entrega dos sistemas por excessivo re-trabalho dos indivíduos e assim por diante. Para uma organização melhorar a qualidade de seus produtos e processos de software, é importante que os programadores, individualmente, disciplinem seus processos pessoais (ZHONG et al., 2000).

O desenvolvimento de software é uma atividade onde a mão-de-obra é intensiva. Nos últimos anos houve a proliferação de métodos e ferramentas, porém, a maior parte dos softwares desenvolvidos é escrita por engenheiros individuais. Então, qualquer melhoria na eficiência ou produtividade de indivíduos resultará em ganhos globais para projetos e, a indústria como um todo (HOU & TOMAYKO, 1998).

A falta de dados sobre as tarefas realizadas, a inexistência de dados históricos sobre o desempenho pessoal do engenheiro de software, são alguns motivos que podem ser levados em conta para utilizar o PSP. Outros motivos são: planejamento mal feito,

não cumprimento de prazos, bem como o aumento do custo do projeto em relação ao orçamento inicial (HUMPHREY, 1995).

A utilização de dados históricos por parte dos engenheiros pode tornar seu trabalho muito mais previsível. A melhor maneira de entender o que está acontecendo de errado em um projeto, é registrando todos os detalhes do processo e, isto é provido pela utilização do PSP. Os dados de projetos prévios mostram que os engenheiros alcançam reduções de 75% no número de defeitos injetados, com a utilização de planos mais precisos. E o mais importante, é que isto ocorre sem redução de produtividade (HUMPHREY, 1995).

O trabalho do engenheiro de software é entregar produtos de alta qualidade dentro do custo e cronograma planejado. Existem três pontos importantes que devem ser respeitados na engenharia de software: elaborar produtos de qualidade, fazer o trabalho dentro do custo esperado e completar o trabalho dentro do prazo. Para fazer um bom trabalho, os engenheiros de software precisam planejar seu trabalho, executá-lo de acordo com o plano e se esforçar para oferecer produtos de alta qualidade.

Uma das tarefas do engenheiro de software é produzir grandes sistemas ou parte deles. Independente do tamanho do sistema ou da parte que se desenvolve, um defeito qualquer pode danificar o sistema por inteiro. O PSP provê ao engenheiro de software um conjunto de benefícios para auxiliar a apresentar produtos de software de qualidade, independentemente do tamanho do sistema.

2.7 BENEFÍCIOS DO PSP PARA A ORGANIZAÇÃO

O PSP provê uma série de benefícios para a organização que o utiliza. Estes benefícios são apresentados a seguir (HUMPHREY, 1995):

Os dados pessoais e de projeto providos pelo PSP auxiliam na elaboração do planejamento e gerenciamento de projeto de software. Estes dados dão fundamentação para as decisões que precisam ser tomadas pelo engenheiro de software na elaboração do plano de projeto.

A utilização do PSP faz com que o engenheiro de software produza softwares com menos defeitos através da utilização da revisão e remoção destes nas fases iniciais do

projeto. Isto também reduz o custo dos testes e pode diminuir o tempo de desenvolvimento.

O PSP utiliza um processo definido de desenvolvimento, através do qual são coletados dados pessoais para gerenciar, controlar e melhorar o trabalho do engenheiro de software. Esta coleta de dados proporciona ao engenheiro de software condições para conhecer seu trabalho. Também é feito o controle das atividades do engenheiro, permitindo a este, saber onde está sendo gasto o tempo de desenvolvimento.

Como o engenheiro passa a ter conhecimento de onde o tempo é gasto, bem como dos defeitos que são injetados no sistema, isto permite ao engenheiro fazer uma avaliação destes dados, possibilitando ao engenheiro fazer a correção de problemas que ocorrem durante o desenvolvimento do software. Com a identificação de todos os erros, é possível ao engenheiro identificar a causa que proporciona estes erros, tornando assim, a solução destes, mais efetiva.

Com o conhecimento de seu trabalho e com a diminuição no número de erros injetados no software, o engenheiro poderá focar sua capacidade para trabalhar em situações que exigem mais criatividade.

O PSP exige do engenheiro de software o planejamento de todos os projetos. Além disso, o PSP faz com que o engenheiro tenha que planejar o seu dia-a-dia, pois estes dados fazem parte do planejamento geral do projeto. O engenheiro precisa saber quais são as tarefas que ele tem pra executar, para conhecer o tempo que terá disponível para trabalhar no projeto e aproveitar melhor o seu tempo no projeto. Se este tempo não for considerado, o planejamento do produto poderá ter problemas.

Com a participação efetiva do engenheiro na elaboração do plano do produto, este se sentirá comprometido com tarefas, fazendo com que ele procure sempre cumprir o que foi estabelecido.

Além disso, o PSP apresenta ao engenheiro 12 áreas chaves do processo do Modelo de Maturidade da Capacidade - CMM (KPA). Isso facilita a preparação dos engenheiros a participar na melhoria na organização baseada no CMM.

Um estudo extenso no PSP foi realizado por Hayes e Over em 1997. Este envolveu 298 engenheiros que gastaram mais que 15.000 horas escrevendo mais de 300.000 LOCs e removendo aproximadamente 22.000 defeitos durante o treinamento

PSP. Os resultados informados dão evidência impressionante de suporte para a efetividade do PSP (HAYES & OVER, 1997).

Baseado em projetos completados, a melhoria média em estimativa de tamanho foi um fator de 2.5. A melhora média na estimativa de tempo foi um fator de 1.75. A redução média em densidade de defeitos global foi um fator de 1.5. Isto incluiu uma redução significativa na porcentagem de defeitos que sobrevivem às fases de compilação e teste, enquanto indica que os hábitos dos engenheiros mudaram de tal modo que os defeitos passaram a ser encontrados nas fases iniciais. A mudança na produtividade foi pequena e estatisticamente insignificante entre os níveis do PSP0 e PSP2 (HAYES & OVER, 1997).

Geralmente, os estudantes são receptivos para aprender os conceitos do PSP. Em uma pesquisa anônima, 81% dos estudantes disseram que sentiam que tinham aprendido sobre gerenciamento e planejamento de tempo e que isto seria benéfico para sua vida acadêmica e profissional. Igualmente, 52% disseram que a matéria sobre remoção de defeitos tinha sido benéfica (WILLIAMS, 2000).

Uma das primeiras empresas a utilizar o PSP foi a *Teradyne*. Quando estes fazem algum investimento, é feita uma análise chamada return *on investiment* (retorno de investimento – ROI). Esta análise indicou que a utilização do PSP em dois projetos de 112 KLOC, foi obtido um benefício de aproximadamente US$5,3 milhões de dólares na economia de tempo de engenharia de software (SEI 2002).

Estes dados são relevantes para as organizações pois mostram que a utilização do PSP pelos engenheiros de software aumenta a qualidade dos produtos desenvolvidos sem que ocorra a perda de produtividade.

2.8 BENEFÍCIOS PESSOAIS DA UTILIZAÇÃO DO PSP

O PSP apresenta vários métodos, para que o engenheiro consiga, com a utilização destes, formular técnicas pessoais para melhoramento de qualidade. A utilização do PSP pelos engenheiros de software cria uma expectativa em relação aos resultados que podem ser obtidos. Dentre os benefícios em relação à utilização PSP, pode-se destacar (HUMPHREY, 1995):

- o engenheiro de software tende a entender melhor o que está fazendo, caso seu trabalho seja definido, medido e acompanhado;
- com este conhecimento e experiência, ele pode selecionar métodos e técnicas que melhor se ajustem às tarefas que executa e às suas próprias habilidades;
- ao utilizar práticas bem definidas e níveis elevados de qualidade no seu trabalho, os engenheiros de software se tornam membros produtivos e eficazes nas equipes de desenvolvimento.

Com a utilização do PSP o engenheiro consegue conhecer melhor o seu trabalho, permitindo, a este, tomar decisões baseadas em dados históricos. O PSP também provê um conjunto de formulários para o acompanhamento de todas as fases do projeto, possibilitando ao engenheiro o acompanhamento de todo projeto. Estes formulários contêm dados relevantes para a qualidade do projeto. Por exemplo, o Modelo de Cenário Operacional, que tem como objetivo registrar os dados dos cenários a serem seguidos na utilização do programa. Este modelo assegura que o engenheiro considere todos assuntos significantes do projeto.

As melhorias de qualidade obtidas com a utilização do PSP são o resultado de três aspectos chave: primeiro, localizando todos os defeitos, os engenheiros conseguem identificar os erros que cometem e ficam mais cuidadosos com seu trabalho. Segundo, quando eles analisam os dados de defeito, eles acabam entendendo que o custo de remover os defeitos nas fases iniciais é menor que nas fases mais próximas do final do projeto, assim aplicam de maneira mais efetiva os métodos de procurar e corrigir defeitos. E terceiro, o PSP apresenta várias práticas de qualidade que provam ser efetivas na prevenção de defeitos e na procura e correção eficaz (FERGUSON, 1997).

2.9 OS FORMULÁRIOS DO PSP

O PSP utiliza grande quantidade de formulários. Isto pode ser visto como uma desvantagem, porém, com o decorrer do tempo, é possível perceber que estes são imprescindíveis. Considere a questão sobre a maneira que os formulários podem auxiliar no planejamento do trabalho. Todo trabalho envolve uma seqüência, uma série de passos. Para fazer um trabalho é necessário:

- determinar o que precisa ser feito;
- decidir como fazer isto;
- executar;
- verificar se o que foi feito está correto;
- corrigir qualquer problema;
- entregar o resultado final.

Estes passos devem ser seguidos em qualquer tarefa, inclusive aquelas consideradas mais simples, que são de fácil execução utilização e compreensão. Os formulários do PSP mostram todos os passos a serem seguidos, as informações pertinentes e que precisam ser registradas. Formulários estruturados e definidos geralmente são difíceis de serem produzidos. Estes precisam ser revisados e melhorados periodicamente para garantir que eles continuem a atender as necessidades do projeto.

É particularmente importante assegurar que todos os formulários que dão suporte a projetos sejam desenvolvidos e mantidos de maneira coerente, ou seja, devem ser revisados e atualizados constantemente. Caso isto não ocorra, podem existir informações duplicadas, terminologia inconsistente, falta de informações ou ainda formatos de formulários que causem confusões. O resultado disto pode ser ineficiência e ou erros. Por esta razão, o PSP provê este conjunto de formulários e *scripts* que auxiliam o desenvolvimento de um software.

Por exemplo, um mergulhador executa uma cambalhota com 2-1/2 mortais, enquanto seu treinador registra isto em vídeo. Então juntos, eles assistem ao vídeo e avaliam os pontos fundamentais do mergulho como partida, extensão e quantidade de mortais. Subjetivamente, o mergulho foi perfeito, porém o mergulhador vê que a posição do corpo dele antes de bater na água era defeituosa. O treinador mostra que para corrigir isto ele precisa endireitar os joelhos mais cedo e apontar os dedos dos pés com mais força. O mergulhador tenta novamente (e novamente, e novamente, ...) (DISNEY, 1998).

Neste exemplo acontece um processo de executar-avaliação-mudança. Mas o que isso tem a ver com o PSP? Para um programador motivado, o PSP pode produzir, de maneira virtual o vídeo do processo de desenvolvimento. Os formulários estruturados e os processos do PSP, provêem ao programador dados do processo de desenvolvimento. Além disso, este modelo faz com que o processo de desenvolvimento possa ser visto de

outras maneiras. O PSP provê uma avaliação constante do treinador para o programador (DISNEY, 1998)

Em algumas áreas, as técnicas estatísticas fundamentadas, auxiliam o "treinador" a fazer declarações como "Em mais de 20 projetos você mostrou uma tendência de fazer uma sub-estimativa de 30% no tempo de requerido para desenvolvimento. Baseado em sua estimativa de 200 minutos para este novo projeto, você provavelmente deveria contar com 280 minutos necessários para terminar." Em outras áreas, podem ser apontados os pontos fracos como: "Você fez 181 erros para cada mil linhas de código. Destes, 64% foram injetados durante o projeto e 80% deles são removidos na fase de testes" (DISNEY, 1998)

Embora o PSP tenha sido desenvolvido para prover orientação e um processo estruturado para desenvolvimento de software, o engenheiro de software é o responsável por encontrar as causas das fraquezas e tomar as decisões necessárias para a correção dos problemas, de modo que o próximo projeto obtenha melhores resultados.

A utilização dos formulários do PSP auxilia os engenheiros de software a desenvolver uma compreensão melhor das fases de desenvolvimento de programa. Os formulários são uma lembrança constante aos engenheiros dos próprios passos a seguir ao desenvolver um programa (GROVE, 1998).

2.10 OS NÍVEIS DO PSP

A implantação do PSP é definida em sete níveis e feita de maneira incremental. Aos níveis superiores são adicionadas características novas em relação aos níveis já implantados. Isto faz com o que o impacto da mudança no processo do engenheiro, no qual somente ele precisa adaptar novas técnicas às já existentes. Os níveis do PSP são: nível de medição pessoal (PSP0 e PSP0.1), nível de estimativas (PSP1 e PSP1.1), nível de qualidade de projeto pessoal (PSP2 e PSP2.1) e nível de processo cíclico (PSP3).

2.11 CONCLUSÃO

O PSP pode ser uma ferramenta útil ao engenheiro de software, desde que utilizada corretamente. O PSP pode prover ao engenheiro dados que possibilitem a

avaliação do seu processo pessoal de trabalho, fornecendo dados que indicam onde este processo deve ser melhorado. Além disso, o PSP pode prover disciplina ao processo de desenvolvimento de software, fazendo com que sejam oferecidos produtos de software de alta qualidade, atendendo a requisitos explícitos e implícitos, bem como entregando o produto no prazo e dentro do custo previamente acordados.

3 OS NÍVEIS DO PSP

3.1 NÍVEL DE MEDIÇÃO PESSOAL

3.1.1 Introdução

A primeira tarefa na introdução do PSP é a medida pessoal. Estas medidas auxiliam na elaboração e utilização de um processo definido para desenvolvimento de software. Um processo de software usa as diretrizes de gerência e técnicas para aplicação de métodos, ferramentas e pessoas para as tarefas do projeto de software. Um processo definido de software permite ao engenheiro de software, construir cada projeto novo com base nas experiências anteriores suas. Um processo definido auxilia no gerenciamento de grandes projetos e ajuda pequenas equipes e o engenheiro fazer um trabalho sozinho (HUMPHREY, 1995).

Além das medidas pessoais, um passo imprescindível no PSP é o planejamento. Não é possível fazer um gerenciamento efetivo, mesmo de projetos modestos com poucos objetos, sem planejamento. O planejamento é uma habilidade que pode ser aprendida e melhorada com a prática constante. Além disso, pode ajudar a fazer um trabalho melhor no desenvolvimento de software.

O PSP não resolve todos os problemas que os engenheiros de software têm no desenvolvimento de software, porém pode auxiliar e guiá-los estabelecendo práticas disciplinadas que podem ser analisadas e melhoradas. Os dados e métricas, providos pelos formulários do PSP são a base para a engenharia de software ser elaborada de maneira científica. Estas métricas também encorajam e apóiam um novo paradigma, mais efetivo para educação, que substitui o professor tradicional de programação por uma pessoa que pode dar deliberação detalhada, específica em como os estudantes podem melhorar (HILBURN, 1999).

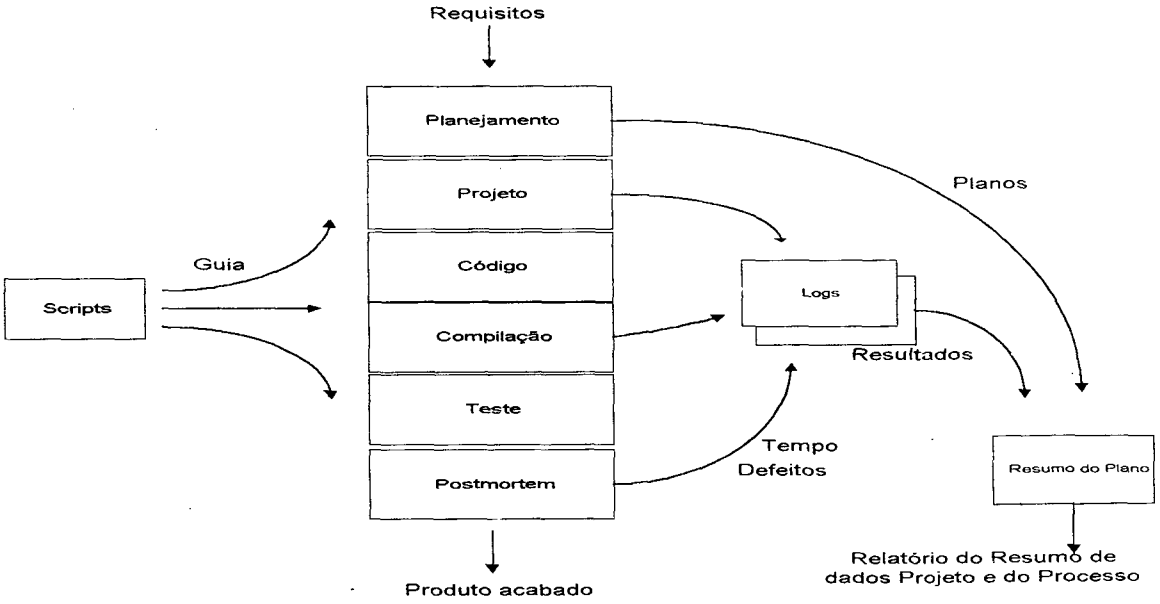
Busca-se aqui, introduzir o processo de medidas pessoais básicas, como tempo de uma atividade, número de defeitos injetados e removidos, bem como faz a introdução de um processo definido para o desenvolvimento de software. Também se faz uma avaliação geral do planejamento do processo. Faz-se uma revisão sobre o porquê do planejamento ser importante e, quais os elementos efetivos de um bom planejamento.

3.1.2 PSP0 – A Base Do Processo

O principal objetivo deste nível é prover uma estrutura inicial para coleta de dados do processo pessoal. Com a coleta de dados do próprio trabalho, é possível construir uma base significativa de informações. Esta base é quem ajuda no entendimento de cada passo do processo e como os métodos do processo podem ajudar na melhora da qualidade e produtividade do engenheiro de software.

No PSP0 as mudanças exigidas são mínimas em relação ao processo do engenheiro de software, apenas são incluídos alguns formulários para a coleta dos dados pessoais. O PSP0 também tem como objetivo mostrar as vantagens de possuir um processo de desenvolvimento de software bem definido. Ele possibilita ao engenheiro de software planejar as tarefas que são executadas e a ordem de execução. O processo do PSP0 é mostrado de uma maneira simplificada na Figura 3.1. Os *scripts* guiam o praticante do PSP nos passos do processo. Os logs auxiliam no registro dos dados e, o resumo do plano provê a maneira correta de registrar e informar os resultados.

Figura 3.1 Fluxograma do Processo do PSP0



Fonte: HUMPHREY, 1995

O processo do PSP0 provê uma estrutura conveniente para fazer as tarefas em pequenas escalas, através do planejamento das tarefas que serão executadas. O que fazer primeiro, segundo e assim por diante, identificando quem executará estas tarefas.

Outra estrutura que o PSP0 provê, é utilizada para medir estas tarefas e é formado por um conjunto de formulários para registro do tempo gasto em cada tarefa, número de defeitos injetados e removidos em cada fase do processo. Estes dados são usados na análise do processo e indicam quais são os erros injetados e onde eles ocorrem, bem como o tempo total gasto em cada uma das fases do processo.

Estes dados proporcionam a fundamentação para melhoria do processo. Há necessidade de saber o que o engenheiro de software está fazendo. Com a estrutura de planejamento e medidas pessoais, isto pode ser verificado, proporcionando assim, a fundamentação para que seja possível fazer as alterações no processo, para que este possa ser melhorado.

Estes elementos de medidas pessoais do PSP0 proporcionam os dados iniciais para acompanhamento do processo de desenvolvimento de software. O engenheiro de software seguindo os *scripts*, formulários e padrões do PSP0, inicia o uso de um processo estruturado de desenvolvimento. Este processo estruturado faz com que o engenheiro trabalhe de maneira organizada, planejando e medindo o seu trabalho.

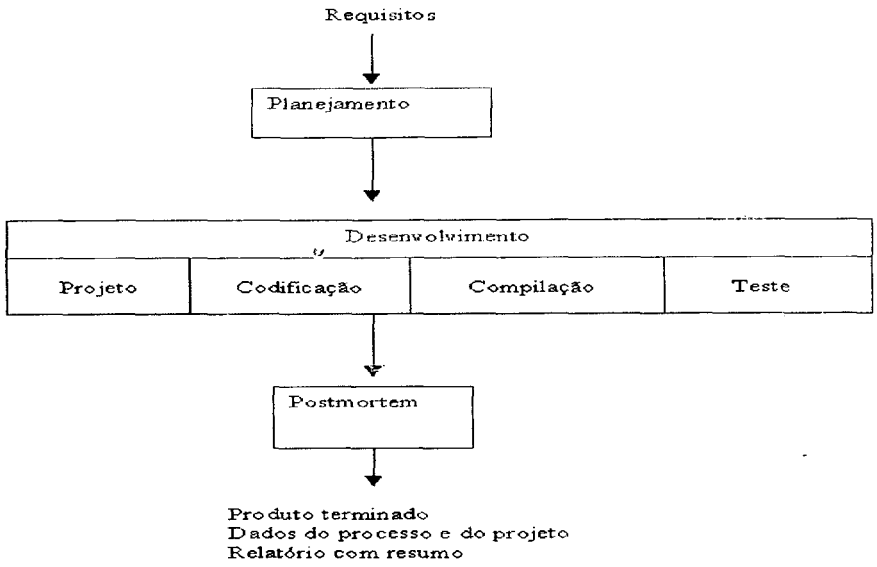
3.1.3 Os elementos do Processo do PSP

O papel dos engenheiros é a produção de soluções econômicas e pontuais, de acordo com as necessidades de seus clientes. Custo e cronograma são dois itens importantes no desenvolvimento de um produto de software e devem ser levados em consideração. Os projetos de software são compromissos e requerem planos. Para que estes possam ser prósperos, os engenheiros de software devem assumir compromissos que possam ser cumpridos e produzir lucros (HUMPHREY, 1995).

O primeiro passo no PSP como mostra a Figura 3.2 é o planejamento, no qual é produzido um plano do trabalho sob a forma de elaboração e execução. Após isto, é desenvolvido o software e, por último, a fase de *postmortem*, na qual é feita uma comparação entre os dados atuais e o planejado. Também é feito o registro dos dados

do processo e produz-se um relatório resumido com as informações pertinentes ao acompanhamento do processo de desenvolvimento.

Figura 3.1 O Processo do PSP



Fonte: HUMPHREY, 1995

Apesar das fases de planejamento e *postmortem* parecerem desnecessárias na produção de pequenos programas, elas são importantes para a construção de um processo disciplinado. É preciso elaborar todas as fases para cada um dos programas escritos. Às vezes o programa parece tão simples que não há necessidade do plano, porém se o programa é simples o plano deste também pode ser simples de se fazer. Freqüentemente, programas simples podem esconder surpresas desagradáveis, tais como, aumento do tempo de elaboração do programa, que podem ser evitados com a elaboração do plano do produto.

3.1.4 O processo do PSP0

Os *scripts* (Processo, Planejamento, Desenvolvimento e *Postmortem*) guiam o praticante do PSP através de um processo de desenvolvimento. Os elementos destes *scripts* são os objetivos, os critérios de entrada, as fases a serem realizadas e os critérios de saída.

O *Script* de processo do PSP0 descreve a estrutura de desenvolvimento e é mostrado no Quadro A.1. O *script* de planejamento é mostrado no Quadro A.2 e resume os passos necessários do planejamento no PSP0. O *script* de desenvolvimento é mostrado no Quadro A.3 e descreve os passos a serem seguidos durante a fase de desenvolvimento e o Script de *Postmortem* resume os passos da fase de *postmortem*, é mostrado no Quadro A.4.

3.1.5 As medidas do PSP0

O PSP0 possui duas medidas: o tempo gasto na execução do plano e o total de defeitos encontrados e removidos em cada fase. O tempo gasto na fase é o registro do tempo em cada fase do processo de desenvolvimento de software. Seu objetivo é determinar onde o tempo está sendo gasto e que alterações são necessárias para melhoria no processo para otimização do trabalho. O total de defeitos injetados e removidos, é o registro do local onde são inseridos e removidos os defeitos. Um defeito é contado cada vez que é feita alguma alteração no programa. Todos os defeitos encontrados devem ser registrados.

A razão de registrar estes dados é formar uma base para planejamento e análise do desempenho pessoal em projetos futuros. Eles fornecem uma linha de base sobre a performance do engenheiro de software, mostram onde é gasto o tempo e onde são injetados e removidos os defeitos. Isto auxilia o engenheiro de software a ver onde o processo deve evoluir e também auxilia a decisão de como mudar o processo para melhorar a produtividade e qualidade do trabalho.

Existem algumas regras para que seja alcançado sucesso com as medidas de processo (ARTHUR, 1994):

- as medidas devem beneficiar as pessoas que coletam os dados;
- o processo deve ser definido, e a partir daí tomar as medidas;
- as medidas do processo devem sempre ser analisadas;
- devem ser definidos reconhecimentos, recompensas para as pessoas que medem.

Alguns princípios úteis a serem adotados na implantação de um processo de medição de software (WEBER et al., 2001):

- as medições devem ser usadas para medir os processo, não as pessoas;
- as medições devem ter objetivos claros e bem definidos;
- o processo de coleta de dados deve ser simples e ferramentas de suporte automatizado devem ser utilizados para a extração dos dados;
- o processo de medição deve ser contínuo e sujeito a melhorias.

3.1.6 Log de Registro de Tempo

A lógica parâ o gerenciamento do tempo é a seguinte: normalmente as pessoas gastam seu tempo da semana da mesma maneira que gastaram na semana que passou. A maneira de gastar o tempo da semana tem uma aproximação muito grande com o modo com que se gasta o tempo nas próximas semanas. Porém, existem exceções, como por exemplo, uma semana na qual o engenheiro participa durante três dias de treinamento, isto sai da rotina de trabalho (HUMPHREY, 1995).

Para fazer planos reais, há a necessidade de registrar onde e de que maneira o tempo é gasto. Algumas pessoas pensam que sabem como gastam seu tempo. Caso façamos uma análise de verificação de onde gastamos o tempo da semana, muito provavelmente, lembraremos de algumas coisas, porém muitas outras são esquecidas.

Para que seja feita uma verificação dos erros cometidos e do tempo gasto no desenvolvimento, é necessário ter isto documentado, para que seja possível fazer a comparação com os dados atuais. Para a realização de bons projetos, é imprescindível fazer planos. A elaboração de planos precisos exige que sejam feitas análises nos planos anteriores para descobrir os erros e o que é preciso fazer para que os planos possam ser melhorados (HUMPHREY, 1997).

Inicialmente, seguir um plano pode parecer difícil. Encontram-se várias razões para não fazê-lo. A mais comum é dizer que o plano não foi bem feito. Planos bons, geralmente, são conseguidos somente após vários planos ruins. O engenheiro de software somente irá conseguir a produção de software de qualidade, quando aceitar seguir um plano de desenvolvimento estruturado e definido. Enquanto isto não ocorrer, a qualidade do produto desenvolvido, sempre será difícil de ser prevista (HUMPHREY, 1995).

O formulário de log de registro de tempo é utilizado para armazenar o tempo das tarefas executadas em cada uma das fases do processo, inclusive interrupções. Um modelo deste formulário é mostrado no Quadro A.5. As instruções para seu preenchimento são indicadas no Quadro A.6. É importante registrar todo tempo gasto no trabalho. A melhor maneira de tornar isto possível, é ter um formulário em branco sempre à mão (HUMPHREY, 1995).

3.1.7 Log de Registro de Defeito

O foco principal de qualidade do PSP está nos defeitos. Para administrar os defeitos, os engenheiros precisam de dados dos defeitos que eles injetam, as fases nas quais foram injetados, removidos e o tempo gasto para os corrigir. Com o PSP, os engenheiros registram os dados de todo defeito achado em qualquer fase, inclusive revisões, inspeções, compilação e testando. Estes dados são registrados no log de registro de defeito (HUMPHREY, 2000).

Um dos primeiros passos no gerenciamento de defeitos é descobrir onde eles ocorrem. Para que isto seja possível, o engenheiro de software deve ter o registro dos dados dos defeitos injetados e removidos. Estes dados ajudam o entendimento deles e torna possível ver o que está sendo feito de maneira equivocada e buscar a maneira de corrigir isto. A melhor maneira de diminuir o número de defeitos em um software é evitar a injeção destes, durante o desenvolvimento (HUMPHREY, 1995).

Para conhecer os dados sobre os defeitos em um programa é preciso fazer o seguinte:

- fazer um registro para cada defeito encontrado em um programa;
- registrar todas informações sobre o defeito para que seja possível entender este mais tarde;
- analisar os dados para ver qual o tipo de defeito que causou a maioria dos problemas;
- buscar maneiras de encontrar e corrigir estes defeitos;
- Atentar para que defeitos não sejam injetados.

O formulário de log de registro de defeito é utilizado para armazenar todos os defeitos encontrados durante o desenvolvimento. Um modelo deste formulário é mostrado no Quadro A.7 e as instruções para preenchimento deste são indicadas no Quadro A.8. É importante registrar todos os defeitos encontrados no processo.

3.1.8 Resumo do Plano de Projeto PSP0

Para que seja possível a confecção de bons planos de projeto há a necessidade de ter as medidas pessoais em um formulário apropriado. Neste, deve ser possível a recuperação de maneira fácil dos dados necessários para os próximos planos. Este formulário é o resumo de plano de projeto. Ao final de cada tarefa, o engenheiro deve completar os dados deste, para poderem ser utilizados em projetos futuros.

O formulário de resumo de plano de projeto contém dados do engenheiro de software, o tempo total planejado, os tempos atuais do processo e o número de defeitos injetados e removidos em cada uma das fases do processo. Este formulário é preenchido na fase de *postmortem*, com os dados oriundos do log de registro de tempo e do log de registro de defeitos. O Quadro A.10 mostra um modelo do resumo de plano de projeto para o PSP0 e o Quadro A.11 contém as instruções para preenchimento. Neste formulário é possível observar de maneira resumida os dados pessoais.

3.1.9 Conteúdo do PSP0

Os *scripts*, formulários, modelos e padrões são os itens usados no processo do PSP. Os *scripts* e o resumo do plano de projeto mudam em cada processo. Porém, o log de registro de tempo, o log de registro de defeito e o tipo padrão de defeitos são usados sem nenhuma alteração nos processos subseqüentes. Todos estes itens são incluídos no Anexo A com as tabelas e números indicados.

Script de Processo PSP0	Quadro A.1
Script de Planejamento PSP0	Quadro A.2
Script de Desenvolvimento PSP0	Quadro A.3
Script de *Postmortem* PSP0	Quadro A.4

Resumo de Plano de Projeto e Instruções	Quadro A.10 e A.11
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

A utilização de todos os formulários, scripts, relatórios e padrões do PSP deve ser seguido pelo engenheiro de software. Inicialmente, o engenheiro pode pensar que são muitas informações para serem registradas, porém somente com estas informações é que será possível a organização do processo de desenvolvimento de software.

3.1.10 Avaliação dos dados do PSP0

Para o aproveitamento da melhor maneira possível dos dados providos pelo PSP, os seguintes itens devem ser observados:

- os dados do processo estão completos.
- os dados são precisos e auto-consistentes.
- o relatório do processo é submetido no formato e ordem própria.

É importante que todos estes critérios sejam atendidos, pois serão usados nas análises das atividades e programas posteriores.

3.2 PSP0.1 PLANEJANDO O PROCESSO DE SOFTWARE

3.2.1 Produzindo um plano de qualidade

O plano é uma definição do trabalho a ser feito e a maneira de fazê-lo. Isto provê uma definição de cada uma das tarefas, estimativa de tempo e recursos necessários e uma estrutura para revisão e controle do processo. Segundo Humphrey (1995), os planos normalmente são utilizados como:

- base para o acordo de custo e cronograma do trabalho;
- estrutura organizada para desenvolvimento do trabalho;
- estrutura para os recursos necessários;
- registro do que foi inicialmente acertado.

Para que se produza um plano de qualidade, Existem, algumas questões chaves que ajudam a verificar as características de um plano de qualidade que são as seguintes (HUMPHREY, 1995):

- Está completo?
- É acessível?
- Está claro?
- É específico?
- É preciso?

Um plano está completo quando cobre todos os objetivos e itens como (estimativas de tamanho, cronograma, orçamento etc) do projeto de software. Nesta situação, a utilização de um processo definido é muito útil para garantir que o plano está completo.

Um plano acessível deve mostrar onde encontrar os dados, deve estar em formato apropriado e não pode ter dados irrelevantes ao processo. Há a necessidade do engenheiro de software saber o conteúdo do plano e onde encontrar isto. A recuperação dos dados de um plano é fundamental para que este possa ser seguido sem problemas.

Os planos devem ser escritos de maneira clara, que qualquer engenheiro de software que o tenha em mãos possa encontrar as informações que necessita. Um plano escrito de maneira obscura, onde alguns dados podem passar despercebidos, pode comprometer todo projeto de software.

Outra característica que o engenheiro de software deve observar na elaboração do plano, é que este seja específico, ou seja, deve conter o que há para ser feito, quando, quem realizará e qual o custo disto.

O plano deve ser preciso. Esta é uma característica importante do plano, pois planos incorretos podem causar atrasos, aumento do custo do projeto, entre outras situações. Para que seja possível fazer planos precisos, o engenheiro de software deve ter os dados que são fornecidos pelos formulários do PSP, estes dados não garantem que o plano seja correto, mas auxiliam a estar próximo da precisão.

Os planos devem ser feitos pelos engenheiros de software com cuidado. Nos planos é que estão incluídos os itens que guiarão os compromissos que serão assumidos para o processo de desenvolvimento de software. O erro que for cometido no plano, poderá causar problemas nas fases posteriores. O engenheiro deve fazer planos em

dados históricos precisos, para aumentar a exatidão do plano. Os planos, geralmente são tarefas difíceis de serem feitas, pois muitas vezes, os dados existentes não são suficientes para a elaboração de planos exatos.

3.2.2 Os planos de produto são úteis

A idéia inicial do PSP é produzir planos para todas as tarefas desenvolvidas. Inclui escrever programas, ler livros texto, preparar relatórios etc. O plano de produto auxilia no acompanhamento das tarefas que são realizadas, bem como, indica quanto tempo se leva para concluir uma tarefa. Os planos também auxiliam no registro do progresso enquanto está se fazendo o trabalho (HUMPHREY, 1995).

Para trabalho em equipe, é útil que os engenheiros de software façam planos de trabalho pessoal. O planejamento provê uma base sólida para as datas de conclusão dos compromissos e também permitem aos engenheiros coordenar seu trabalho junto com os produtos. Os planos de produtos podem ser utilizados para entender o andamento do projeto. Por exemplo, saber em qual fase do projeto se está. Os planos são importantes e, há a necessidade de serem feitos de maneira adequada a cada projeto.

3.2.3 O que é um plano de produto

O plano descreve a maneira pela qual um projeto específico se realiza: como, quando e qual o custo. Os planos também podem ser feitos para tarefas individuais. O primeiro passo para a produção de um plano é ter visão clara do que está se pensando construir. Isto parece óbvio, mas é surpreendente como as pessoas iniciam o desenvolvimento de produtos sem definição clara do que eles desejam fazer (HUMPHREY, 1997).

Após saber o que deve ser feito, é possível pensar o como fazer. Neste momento, inicia-se o planejamento. Conforme Humphrey (1997), para que se tenha um plano de produto apropriado, este deve incluir três coisas:

- tamanho e características importantes do produto a ser produzido;
- estimativa do tempo necessário para fazer o trabalho; e

- projeção do cronograma.

Para que seja possível produzir um software de qualidade, o engenheiro deve fazer uso do plano de produto. Sem a utilização deste, corre-se o risco do projeto não ser concluído, pela falta de organização das tarefas a serem executadas. O plano deve conter de forma clara, o que está sendo construído.

3.2.4 Planejamento de um projeto de software

Para que a elaboração de um produto de software seja concluída com êxito, ou seja, que atinja seus objetivos, há a necessidade de iniciar o trabalho com o planejamento do produto a ser desenvolvido. O Planejamento de um produto de software é uma diretriz do que vai ser produzido e como isto será feito.

Quando da elaboração de produtos que será gasto mais de um dia de trabalho, o trabalho deve ser dividido em tarefas menores, pois pode facilitar a compreensão e execução. É importante levar em consideração os dados de projetos anteriores, para dar fundamentação às estimativas. Assim como é importante fazer a análise das estimativas com os dados atuais do projeto, para verificar o que estava certo e corrigir o que foi elaborado de maneira errônea.

Os passos seguintes auxiliam na construção do processo de estimativa estável e efetivo (HUMPHREY, 1995):

- a tarefa deve ser iniciada com a declaração explícita do trabalho a ser feito e assegurar-se de que é isto que o cliente espera que seja feito. Os modos nos quais projetos podem diferir são infinitos e não há nenhuma estrutura de plano padrão que se ajuste para a maioria dos projetos em uma organização. Assim, é importante refletir para o que é requerido para este projeto particular e como se faz;
- para tarefas que exigem mais de uns dias de trabalho, divida este em várias tarefas pequenas e faça a estimativa de cada tarefa separadamente. Todos os detalhes adicionados melhoram a precisão do plano;
- as estimativas devem ser baseadas em comparações deste trabalho com os dados históricos de projetos anteriores;

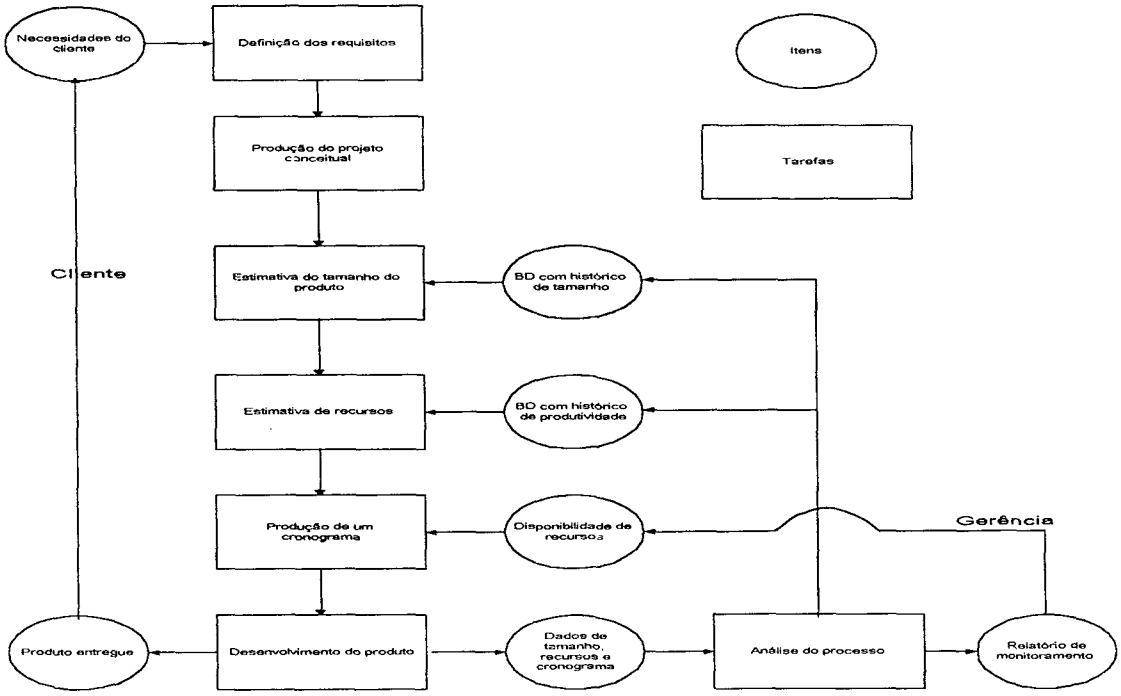
- as estimativas devem ser registradas para que mais tarde possam ser comparadas com os dados atuais do projeto.

3.2.5 Estrutura de planejamento

A Figura 3.3 mostra a estrutura de planejamento do PSP. As tarefas são representadas pelos retângulos, enquanto que os dados, relatórios e produtos são mostrados nas figuras ovais. O processo inicia com a necessidade do cliente, onde são definidos os requisitos. Após isto, produz-se o projeto conceitual, relacionando as estimativas do planejamento com o produto a ser construído.

Com o projeto conceitual e os dados históricos de produtos anteriores, é possível estimar o tamanho do novo produto. Juntamente com a estimativa de tamanho, utilizam-se os dados históricos de produtividade para estimar quantas horas de trabalho são gastas. Após isto, deve-se alocar estas horas em um calendário para que seja possível produzir o cronograma de atividade. A partir do momento em que for definida a data de início é possível calcular a data de término do trabalho.

Figura 3.1 Estrutura de planejamento de Processo



Fonte: HUMPHREY, 1995

Com o plano em mãos é possível dar início ao desenvolvimento. Durante o desenvolvimento e conclusão do projeto devem ser registrados o tempo gasto e o tamanho do produto produzido. Estes dados serão utilizados na elaboração de relatórios e análise do processo. Estas análises irão prover os dados de tamanho e produtividade para planos futuros.

3.2.6 Medidas de tamanho

A principal diferença entre o PSP0 e o PSP0.1 é a adição de medidas de tamanho utilizando *Lines of Code* (Linhas de código) - LOC com unidade de medida. O PSP0.1 introduz também o *Process Improvement Proposal* (Proposta de Melhoria do Processo) - PIP que é utilizado para registrar idéias sobre melhorias no processo, lições aprendidas, soluções para problemas ou outras sugestões. Também inclui um padrão de codificação mostrado no Quadro A.81. Este, pode ser modificado pelo usuário para atender necessidades específicas, preferências e linguagens de programação.

Estas mudanças significam que na fase de planejamento o usuário faz uma estimativa do total de linhas necessárias, novas e modificadas. Adicionalmente, desde que existam projetos anteriores, deve-se distribuir o total de tempo planejado entre as fases de desenvolvimento. Desta forma, deve existir uma quantidade planejada de tempo para cada fase do processo.

Na codificação, devem ser utilizados os padrões de codificação desenvolvidos. Ao longo do desenvolvimento, o engenheiro de software pode anotar idéias ou melhorias no formulário PIP a qualquer momento. Em algum momento do desenvolvimento do software, o engenheiro pode ter uma idéia para melhoria do processo. Porém, se esta não for anotada na hora, ela pode ser esquecida. Por isso, é importante ter sempre junto um formulário PIP para anotações.

Na fase de *postmortem* o engenheiro mede ou calcula as LOC do programa atual, pronto para oito categorias: tamanho básico do programa, apagadas, modificadas, adicionadas, reutilizadas, novas e modificadas, total de LOC e total de novas reutilizáveis. A partir destes dados o usuário consegue responder as seguintes perguntas:

como este projeto se mostrou para ser comparado com minha estimativa inicial? Ou, quantos códigos foi possível reutilizar neste novo projeto?

O processo de planejamento de software inicia com a estimativa de tamanho do trabalho. Antes de estimar o tamanho do software, porém, é necessária uma maneira consistente e repetível para descrever o tamanho de um produto. Na seleção de medidas de tamanho de software deve-se assegurar que as medidas selecionadas são úteis e precisas para o planejamento. Isto é, quando várias pessoas independentes medirem o mesmo produto, os resultados obtidos devem ser idênticos.

Os dois principais critérios para a realização de medições em software são os seguintes (HUMPHREY, 1995):

- comunicação: se alguém usar estes mesmos métodos para definir uma medida ou descrever o resultado de uma medida, outros sabem precisamente o que foi medido, o que foi incluído e o que foi deixado de fora?
- repetível: outra pessoa pode ser habilitada para repetir a mesma medida e o mesmo resultado?

Com base nestes critérios o SEI estabelece uma estrutura para definição precisamente métrica de LOCs. Uma versão do formulário é apresentada no Quadro A.79 e as instruções no Quadro A.80. Considerando a grande margem de erro da contagem LOC, o engenheiro de software deve tratá-la com muito cuidado.

Ao medir a produtividade do desenvolvimento, os engenheiros normalmente contam o número de linhas de código por hora de desenvolvimento. Para este propósito, deve-se contar os desenvolvimentos recentes juntamente com as declarações modificadas. Também é importante usar precisamente as mesmas definições ao se estimar a produtividade de desenvolvimento e planejamento de projeto.

Para a verificação da taxa de defeitos por programas, é comum utilizar a taxa de defeitos por mil linhas de código adicionado e modificado. Ao estimar a carga provável de trabalho para manutenção de um programa, é mais apropriado considerar as LOCs totais do produto terminado. Um contador de LOCs pode ser projetado para contar linhas físicas ou linhas lógicas. Um terceiro método combina estes dois: conta linhas lógicas usando um padrão de codificação e um contador de LOCs físicas. Esta é a abordagem utilizada no PSP.

Podem ser obtidas muitas estatísticas de LOCs úteis com ferramentas corretamente projetadas. Para cada codificação do programa, o engenheiro pode querer saber quantas LOCs foram adicionadas e apagadas. Um modo prático de obter tais dados é com um programa que compare cada versão do programa com a versão anterior. Este comparador então identifica e conta cada linha adicionada ou apagada. A chave para medidas de tamanho de software efetivas é assegurar que elas se ajustem às necessidades pessoais do engenheiro de software (HUMPHREY, 1995).

3.2.7 Conteúdo do processo

O PSP0.1 apresenta medidas de tamanho, o formulário PIP (Proposta de Melhoria do Processo) e o padrão de codificação. Antes de utilizar o PSP0.1 para desenvolver um programa deve ser feita uma estimativa do tamanho e completar a estimativa de tempo no Resumo de Plano de Projeto. Ao término do desenvolvimento deve-se medir o programa, calculando o tamanho básico do programa, apagadas, modificadas, adicionadas, reutilizadas, novas e modificadas, total de LOC e total de novas reutilizáveis.

Os novos elementos do processo incluem não somente scripts e relatório de resumo, mas também o *Process Improvement Proposal* (Proposta de Melhoria do Processo) PIP e o Padrão de Código. Todos estes itens são incluídos no Anexo A com todos os quadros e números indicados.

***Script* de Processo PSP0.1**	**Quadro A.12**
***Script* de Planejamento PSP0.1**	**Quadro A.13**
***Script* de Desenvolvimento PSP0.1**	**Quadro A.14**
Script de *Postmortem* PSP0.1	**Quadro A.15**
Resumo de Plano de Projeto PSP0.1 e Instruções	**Quadro A.16 e A.17**
PIP Proposta de Melhoria do Processo e Instruções	**Quadro A.18 e A.19**
Padrão de código	**Quadro A.81**
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

3.2.8 Novos elementos do processo

O *Process Improvement Proposal* (Proposta de Melhoria do Processo) PIP é usado para registrar todos os problemas ou sugestões de melhoria que o engenheiro de software tiver durante o processo de desenvolvimento. Muitos problemas na definição de processo concernem de detalhes secundários. Porém estes detalhes fazem a diferença entre um processo inconveniente e incômodo de um processo confortável e eficiente. Por este motivo há a necessidade de registrar todos estes detalhes (HUMPHREY, 1995).

Em algumas situações, durante uma fase do processo ou um formulário, pode surgir um problema e as dificuldades para lembrar dos detalhes mais tarde são maiores. Um formulário PIP em branco deve sempre ser mantido à mão para registro de qualquer idéia de melhora do processo. Surpreendentemente, se a idéia não for registrada na hora, esta, pode ser difícil de reconstruir mais tarde. Os detalhes são importantes para o processo pessoal e para corrigir estes detalhes, deve-se regularmente completar e manter o formulário PIP.

O formulário PIP pode ser usado para fazer sugestões de melhorias do processo. Pode também registrar o que aconteceu durante o processo, particularidades que possam ser interessantes ou úteis. Anota-se ainda, qualquer problema ocasional que ocorra durante o desenvolvimento. O formulário PIP é mostrado no Anexo A, Quadro A.18. O engenheiro de software deve ter um formulário desses sempre à mão.

O padrão de código proposto por Humphrey (1995) é mostrado no Anexo A, Quadro A.81. Enquanto os programas devem ser corretos, o texto do programa também chamado de código fonte, deve ser compreensível. Escrever um código fonte fácil de ler, ajuda a pensar mais claramente sobre o projeto, teste, modificação e reutilização. Comentários objetivos e precisos auxiliam o entendimento do programa, até mesmo quando não se espera que seu programa seja usado por outra pessoa. Deve-se criar o hábito de escrever códigos legíveis (HUMPHREY, 1995).

O padrão de código é útil tanto em listagem de programas grandes como também em programas menores. Para ser mais útil, o padrão de código deve ser desenvolvido para a linguagem e ambiente de trabalho em uso. Deve-se adaptar o padrão de código de acordo com as necessidades de cada ambiente. Ao desenvolver um padrão de código

todos os engenheiros de software devem ter conhecimento sobre todos os itens de programação incluídos e excluídos deste.

3.2.9 Avaliação dos dados do PSP0.1

Para o aproveitamento da melhor maneira possível dos dados providos pelo PSP, os seguintes itens devem ser observados:

- os dados do processo estão completos.
- os dados são precisos e auto-consistentes.
- o relatório do processo é submetido no formato e ordem própria.

É importante que todos estes critérios sejam atendidos, pois serão usados nas análises das atividades e programas posteriores.

3.3 NÍVEL DE ESTIMATIVAS

3.3.1 PSP1 - Estimativa De Tamanho

Um problema que se apresenta na engenharia de software é o processo de estimar tamanho de produtos de software. Em algumas situações, na fase onde a estimativa é exigida, o engenheiro não consegue identificar todas as características do produto que está para ser estimado.

O PSP1 apresenta o processo de estimativas de tamanho e tempo e descreve o método de estimativa PROBE (*PROxy Based Estimating*), desenvolvido por Watts Humphrey como um subproduto do PSP. A idéia principal é estimar o tamanho do software a ser desenvolvido, a partir do tamanho dos objetos identificados como seus possíveis componentes.

Uma razão para a estimativa de tamanho de um produto de software, é auxiliar no desenvolvimento do cronograma de entrega do produto e para ser usado na elaboração do plano do projeto. A qualidade de um plano de desenvolvimento de software, geralmente depende da qualidade da estimativa de tamanho.

O plano do produto contém dados sobre o trabalho a ser feito, quando e quem executará o trabalho. Através das estimativas, é possível prever quanto tempo será gasto na tarefa. A situação onde o planejamento de software possui erros, é uma das

principais razões dos projetos de software terem problemas nas fases posteriores do projeto. Causa freqüente de planos ineficazes são estimativas de tamanho ineficazes (HUMPHREY, 1995).

O grau de precisão do plano de trabalho depende do quanto é conhecido do produto a ser produzido. No início, normalmente, somente tem-se uma idéia geral dos requisitos do produto. Nesta situação a única maneira de uma estimativa é através de uma analogia a produtos feitos anteriormente. Após a fase inicial, cada vez mais, vai se sabendo sobre o produto a ser desenvolvido.

Sempre que for preciso fazer estimativas, é necessário definir o projeto no maior número de detalhes possível. As estimativas para grandes projetos de software podem ser controladas dessa maneira: deve-se iniciar com uma especificação do projeto, e depois examinar e estimar cada parte do projeto (HUMPHREY, 1995).

Este trabalho requer estimativas separadas, para cada componente do software, casos de teste, planejamento de instalação, conversão de arquivos e treinamento de usuários. As telas, relatórios ou lógica funcional também devem ser calculadas. Para grandes produtos de software, existe um grande número de detalhes potenciais que precisam ser considerados no momento de fazer a estimativa (HUMPHREY, 1995).

O engenheiro de software precisa estar atento, pois uma falha no levantamento dos detalhes e requisitos do produto, pode ocasionar problemas nas estimativas e no planejamento do produto. Esta situação pode ocasionar atraso na entrega do produto, bem como, aumento do custo do produto em relação ao valor inicial estipulado.

3.3.2 Critérios de estimativa de tamanho

O objetivo do PSP1 é estabelecer um procedimento ordenado e repetível para as estimativas de tamanho de desenvolvimento de software. Isto faz com que o engenheiro de software consiga fazer a estimativa para projetos de software de qualquer tamanho.

Os métodos de estimativa de tamanho devem satisfazer aos seguintes critérios (HUMPHREY, 1995):

- usar métodos estruturados e treináveis. Métodos estruturados facilitam o treinamento e a melhoria do processo. Isto permite acompanhar e melhorar seu próprio método de estimativa;

- deve ser possível utilizar este método durante todas as fases de desenvolvimento e manutenção do software. É preciso usar as estimativas de tamanho desde o início do ciclo de desenvolvimento para fazer planos e compromissos reais. Durante o desenvolvimento podem ser necessários ajustes de acordo com as alterações do projeto;
- deve ser usado por todos os elementos do produto de software. É preciso controlar não somente o código, mas, arquivos, relatórios, telas e documentação;
- deve satisfazer as análises estatísticas. Métodos de estimativa de tamanho precisam prover os meios para os ajustes nos parâmetros de estimativas com base em dados históricos;
- deve ser adaptável aos tipos de trabalho que podem ser feitos no futuro. Assim que se obtém dados de estimativas e experiência, também se adquire um recurso de valor contínuo;
- deve prover os meios para julgar e analisar a precisão das estimativas.

Ao final de cada trabalho, o engenheiro de software deve comparar as estimativas com os dados atuais do produto. Na análise destas comparações é possível identificar os erros de estimativa que ocorreram e as suas causas. Isto permite fazer ajustes necessários para melhorar o método.

No planejamento de projeto, as estimativas são geralmente necessárias antes do início do desenvolvimento. No estágio inicial, os requisitos podem ser entendidos, porém pouco é sabido sobre o produto em si. O problema da estimativa é prever o tamanho final do produto a ser produzido. Como ninguém pode dizer exatamente o tamanho de um produto planejado, a estimativa de tamanho tem sempre uma incerteza. É necessário fazer a estimativa tão precisa quanto possível. Em geral os métodos de estimativas usam dados de programas similares feitos anteriormente para base de calculo do tamanho do novo produto (HUMPHREY, 1995).

3.3.3 Estimativa baseada em substituto

Se fosse possível estimar o número de LOCs com precisão em um produto de software planejado, as estimativas de tamanho e tempo de desenvolvimento seriam

feitas com precisão. Porém, é muito difícil julgar quantas LOCs são necessárias para satisfazer os requisitos de um software. A comparação entre produtos de software é uma tarefa difícil de ser realizada. Devido a isto, o SEI desenvolveu um método para estimativa de tamanho baseado em *proxy* (substituto).

A necessidade é para algum *proxy* que relacione o tamanho de produto às funções do produto, fazendo com que o engenheiro de software possa visualizar e descrever as características do produto. Um *proxy* é um substituto e pode ajudar a julgar o tamanho de um produto. Exemplos de *proxy* são objetos, telas, arquivos, *scripts*, ou pontos de função (HUMPHREY, 1995).

Os critérios para selecionar o *proxy* são os seguintes (HUMPHREY, 1995):

- a medida de tamanho do *proxy* deve se relacionar com o esforço necessário para desenvolver o produto;
- o conteúdo do *proxy* de um produto deve ser automaticamente contável;
- o *proxy* deve ser fácil de visualizar no início do projeto;
- o *proxy* deve ser personalizado às necessidades particulares da organização;
- o *proxy* precisa ser sensível a qualquer variação de implementação que cause impacto no custo ou esforço de desenvolvimento.

O *proxy* pode ajudar o engenheiro a fazer a estimativa de tamanho. O *proxy* é um componente de um produto já desenvolvido, que será usado como substituto, na elaboração da estimativa de tamanho de um produto novo. Fazer estimativas, em algumas situações, é difícil pela falta de conhecimento do produto. Quando um objeto (arquivo, tela etc) é identificado, há a necessidade de saber quantas LOCs serão escritas para este objeto. A utilização de *proxy* como substituto, permite que o engenheiro faça comparações, entre o *proxy* e o novo objeto, sendo assim, base para as estimativas.

3.3.4 O método PROBE (*PROxy-Based Estimating*)

O método de estimativa PROBE (*PRoxy Based Estimating*) foi criado por Watts Humphrey no SEI (HUMPHREY,1995), como subproduto do desenvolvimento do PSP. A sua proposta é produzir uma estimativa de tamanho de um software em LOC no início de seu ciclo de desenvolvimento, conforme exigido por clientes e usuários. A

idéia principal é estimar o tamanho do software a partir do tamanho dos objetos identificados como seus possíveis componentes.

O problema é que no momento inicial, as informações disponíveis aos engenheiros de software são as extraídas de uma descrição de alto nível (muitas vezes vaga) de requisitos. Em algumas situações nem o cliente tem idéia exata do produto que deseja. As estimativas produzidas através destas descrições, mesmo considerando a existência de informações históricas gerenciadas para comparações, é uma atividade difícil de ser executada (HUMPHREY, 1995).

Mesmo que os requisitos sejam detalhados, estes, apenas fazem a descrição do produto final e não indicam como o produto será construído. Um dos motivos é a dificuldade em encontrar similaridades suficientes em projetos prévios para que seja possível a comparação.

Partindo desta situação, o projeto detalhado do sistema deve ser feito antes de qualquer estimativa.Porém considera-se que o projeto detalhado do sistema é parte do ciclo de desenvolvimento, logo, submetido a prazos em função de custo e expectativas. Assim, todo ciclo de desenvolvimento do produto deve ser estimado, e não apenas a construção física do produto.

Uma tentativa para resolver esta situação é fazer o que o método recomenda, um projeto conceitual, com base em um julgamento breve de como o produto é construído. A partir deste projeto conceitual, compara-se este modelo com os dados históricos na base de informações e estima-se o tamanho do produto com base em projetos prévios. Um problema que pode ocorrer é a situação onde nenhum projeto semelhante é encontrado na base de dados. Para resolver esta situação, podem ser feitas analogias, considerando projetos maiores e menores e estabelecer uma faixa de tamanho onde a nova proposta se encaixe.

Nesta situação não está considerado um problema citado anteriormente, que é a dificuldade de comparação entre dois produtos de software. Mesmo quando os programas são pequenos, podem existir muitas diferenças e variações, difíceis de serem vistas neste nível de abstração. Quando se está trabalhando com programas grandes, é praticamente impossível executar esta comparação de maneira consistente. Isto se dá pela complexidade e o número de funções em produtos grandes é muito maior, para que seja possível atender a todos os requisitos.

Uma solução para esta situação é a divisão do software em partes menores. A comparação de partes de um software pode ser mais confiável do que a comparação de produtos inteiros. Partes similares podem ser utilizadas para resolver problemas diferentes. É possível determinar possíveis reutilizações de partes já prontas, ou que exigem adaptações de baixo impacto.

O método PROBE trata o problema da estimativa de tamanho das partes identificadas não pré-construídas pela comparação não com partes idênticas, mas com tipos de partes similares. Esta medida estimada, conforme sugerido pelo seu criador, é em LOC, considerando a simplicidade de seu uso e a facilidade de controle. O método utiliza objetos como substitutos, cuja presença pode ser detectada já no projeto conceitual devido ao fato de representarem coisas do mundo real incorporadas ao software (PRANTONI, 2002).

Como exemplo análogo, determinada pessoa solicita a um arquiteto que lhe construa uma casa e, deseja uma estimativa de prazos e custos para a obra. O arquiteto, para isso, precisa conhecer detalhes precisos da obra, ou seja, um projeto completo e, particularmente a metragem quadrada desejada. O cliente não sabe determinar esta medida, mas sabe que quer um quarto "grande", dois "médios", cozinha, dois banheiros, sala... O arquiteto pode utilizar informações de construções que contemplam estas solicitações do cliente e determinar pelo menos uma faixa de valores. O cliente deve avaliar se está de acordo com suas expectativas.

Neste exemplo, os cômodos são substitutos para metros quadrados. O método PROBE usa objetos como substitutos para linhas de código. Estima-se, a partir do modelo conceitual, o número de objetos de determinado tipo necessário para o produto. Atribui-se a faixa de linhas de código pré-estabelecida para cada tipo de objeto, em função dos dados históricos. Finalmente, determina-se a faixa de tamanho estimada para o produto e calcula-se o tempo necessário para sua construção.

Objetos são interessantes como substitutos pelos seguintes motivos (HUMPHREY, 1995):

- são fáceis de visualizar no início do processo de desenvolvimento;
- normalmente, na fase de análise, classes abstraem também as entidades da aplicação, portanto são partes bem definidas que envolvem os aspectos dinâmicos e estáticos do modelo;

- podem ser automaticamente contados por ferramentas adequadas;
- sua classificação (tipo) pode ser adequada às necessidades de cada organização ou engenheiro de software.

A maior complexidade do método está na criação de seu *framework*, ou seja, como determinar os tipos de objetos por tamanho e funcionalidade, que são utilizados como base para estimativas e, aprimorado com a evolução dos dados históricos. Não se pode deixar de considerar o fato de que um mesmo problema pode ser tratado de maneiras totalmente diversas por engenheiros de software diferentes. Eles variam bastante o número de linhas de código produzidas e o tempo necessário para o seu desenvolvimento.

Como o PROBE foi desenvolvido como um subproduto do PSP, ele prevê que cada engenheiro tenha a sua base de dados pessoal. Cálculos estatísticos podem ser utilizados para se obter um padrão para a organização. Humphrey classificou objetos C++ em seis tipos: cálculo, dados, E/S, lógico, *setup* e texto. Estabeleceu faixas de tamanho para cada tipo de objetos. Calculou o desvio padrão do número de linhas de código dos métodos para obter a estimativa de cada tipo por faixa (Humphrey utilizou as faixas "muito pequeno", "pequeno", "médio", "grande" e "muito grande") (HUMPHREY, 1995).

A primeira etapa da aplicação do método é a decomposição do modelo conceitual, produzido por qualquer método orientado a objetos, em partes componentes. Segue-se a repetição do processo até que todos os componentes sejam passíveis de comparação com componentes já produzidos e enquadrados em tipos definidos no *framework*. Observa-se que este é apenas um projeto conceitual desenvolvido para identificar partes da futura aplicação e não determinar como elas são construídas.

Atingido o nível de detalhamento necessário, deve-se observar cada objeto identificado, decidir o seu tipo e estimar seu número de métodos. É importante nomear os métodos, para assegurar a identificação da funcionalidade do objeto. Com isto, diminui a probabilidade de erros de estimativa identificando corretamente o seu tipo ou a necessidade de um detalhamento maior desta área do modelo.

Identificados os tipos e número de métodos de cada objeto, deve-se identificar os objetos de mesmo tipo e faixa de tamanho que existem na base de informações

históricas. Quando não há certeza quanto à faixa de tamanho, a recomendação é de classificá-lo como médio.

Eventualmente, pode ser necessário estimar sem comparações mais consistentes. Isto pode ocorrer pela inexistência de objetos de um determinado tipo na base de informações históricas. Outra razão é porque determinados objetos do modelo possuem métodos de tipos muito variados impossibilitando o ajuste em algum tipo determinado no *framework* ou mesmo a sua categorização. Neste caso, a única alternativa possível é o bom senso e a experiência pessoal do engenheiro de software.

Depois, é necessário atribuir o número de LOC calculado por método no *framework* para cada objeto do modelo. Após isto, multiplica-se pelo número de métodos do objeto para obter o total de LOC do mesmo e somar os totais obtidos para chegar ao total de LOC estimado de todo o projeto (PRANTONI, 2002).

Para exemplo, considera-se a estimativa que um estudante do PSP[1] fez para o programa 10A utilizando a linguagem C++, que é mostrado a seguir no Quadro 4.1.

Quadro 3.1 Exemplo de Estimativa de Tamanho

Estudante: **Estudante 12**				Data: **05/01/1994**	
Instrutor: **Humphrey**				Programa **10A**	
LOC base do programa				Estimado	Atual
Tamanho base (B) => => => => => =>				**695**	**695**
LOC deletadas (D) => => => => => =>				**0**	**0**
LOC modificadas (M) => => => => => =>				**5**	**18**
Objetos LOC					
Adições da Base	Tipo	Métodos	Tam. Rel.	LOC	LOC
Total de adições da base (BA) => => => => => => =>					
Novos objetos	Tipo	Métodos	Tam. Rel.	LOC (novas reutilizáveis*)	
Matrix	**Date**	**13**	**Médium**	**115**	**136**

[1] Estimativa do estudante 12 do livro de Humphrey pg. 120 (HUMPHREY 1995)

Linear System	**Calc.**	**8**	**Large**	**197**	**226**
Linked List	**Data**	**2**	**Large**	**49***	**54***
Control	**Logic**	**2**			**48**
Total de novos objetos (NO) => => => => => => =>				**361**	**464**
Objetos reutilizados					
Linked List				**73**	**73**
Data Entry				**96**	**96**
Total de objetos reutilizados (R) => => => => => =>				**169**	**169**
				Tamanho	Tempo
LOC Estimada dos objetos	E = BA + NO + M			**366**	
Parâmetros de regressão:	β0 (tamanho e tempo)			**62**	**108**
Parâmetros de regressão:	β1 (tamanho e tempo)			**1,3**	**2,95**
LOC estimadas novas e modificadas (N)	N = β0 + β1 * E			**538**	
LOC total estimadas	T = N + B – D – M + R			**1397**	
Total estimadas novas reutilizáveis (soma das LOC)				**49**	
Tempo estimado desenvolvimento Tempo = β0 + β1 * E					**1186**
Limites de predição	Limite			**235**	**431**
Limite superior de predição	UPI = N + Limite			**773**	**1617**
Limite inferior de predição	LPI = N – Limite			**303**	**755**
Percentual do intervalo de predição				**90%**	**90%**

Fonte: HUMPHREY, 1995

Usando o projeto conceitual, o estudante nomeia cada novo objeto e determina seu tipo. O primeiro objeto, *Matrix* é do tipo dados. O estudante estima quantos métodos ou procedimentos este objeto pode ter, neste caso 13. Depois disso, esse objeto tem seu tamanho relativo estimado como médio. Isto é determinado a partir da base de dados históricos para objetos C++, onde objetos do tipo dados e tamanhos relativos médios têm 8,84 LOC por método. Multiplicando este valor pelo número de métodos do objeto, neste caso, 13, obtendo 114,9 LOC. Estes mesmos passos são repetidos para os outros objetos, chega-se a um total de 361 LOC de objeto novas (HUMPHREY, 1995).

O script de estimativa PROBE, Anexo A, Quadro A.26, descreve como obter os outros cálculos para as estimativas.

O tempo necessário para fazer as estimativas, está intimamente relacionado ao detalhamento do modelo conceitual e a precisão da estimativa está ligada ao aprofundamento de detalhes.

3.3.5 Conteúdos do PSP1

O PSP1 introduz a estimativa de tamanho ao PSP. Antes de iniciar um desenvolvimento de software do PSP1, deve-se utilizar o método PROBE para fazer a estimativa do tamanho de LOC novas e modificadas, total da base, reutilizadas, apagadas, modificadas, adicionadas, o total de novas modificadas e novas reutilizáveis.

Deve-se observar que na utilização do PSP pela primeira vez, não existem dados históricos para fazer a estimativa de tamanho do produto. Logo, para estimar as LOC dos objetos é necessário usar o próprio julgamento para calcular o total de LOC novas e modificadas. Na elaboração dos primeiros programas, os dados destes são adicionados à base de dados, assim, é possível utilizá-los para as estimativas dos programas seguintes.

Todos estes itens estão inclusos no Anexo A com todas as tabelas e números indicados.

***Script* de Processo PSP1**	**Quadro A.20**
***Script* de Planejamento PSP1**	**Quadro A.21**
***Script* de Desenvolvimento PSP1**	**Quadro A.22**
***Script* de *Postmortem* PSP1**	**Quadro A.23**
Resumo de Plano de Projeto PSP1 e Instruções	**Quadro A.24 e A.25**
***Script* de Estimativa PROBE**	**Quadro A.26**
Modelo de Relatório de Teste e Instruções	**Quadro A.29 e A.30**
Modelo de Estimativa de Tamanho e Instruções	**Quadro A.31 e A.32**
PIP Proposta de Melhoria do Processo e Instruções	Quadro A.19
Padrão de código	Quadro A.81
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

3.3.6 Novos elementos do processo

O Modelo de Estimativa de Tamanho foi descrito anteriormente e está no Quadro A.31. Este modelo é utilizado para registro dos dados sobre as estimativas de tamanho dos produtos a serem desenvolvidos.

O Modelo de Relatório de Teste é mostrado no Quadro A.29, é usado para registrar os dados de cada um dos testes executados. As condições de teste e resultados podem parecer sem importância para pequenos programas, porém quando estes são incorporados em grandes programas, alterações posteriores, correção de problemas ou até mesmo na reutilização de programas, são muito úteis na re-execução dos testes. Para que isto seja possível, é necessário conhecer os dados de cada teste executado, quais dados foram utilizados e os resultados obtidos.

Os dados sobre os testes executados são úteis quando da modificação do programa ou quando for incorporada a uma nova função. Como parte do teste, é necessário assegurar que as modificações não causem problemas nas funções previamente usadas. Esses problemas são chamados de regressão. Os problemas de regressão são comuns quando muitas alterações são feitas em versões do programa. Um teste de regressão será fácil de ser executado, quando executados novamente os testes previamente registrados. Porém, isto se torna muito difícil, quando os dados anteriores não são registrados. Estes dados são fáceis de registrar quando são feitos junto ao teste, porém são difíceis de reconstruí-los depois.

3.3.7 Avaliação dos dados do PSP1

Para o aproveitamento da melhor maneira possível dos dados providos pelo PSP, os seguintes itens devem ser observados:

- os dados do processo estão completos.
- os dados são precisos e auto-consistentes.
- o relatório do processo é submetido no formato e ordem própria.

É importante que todos estes critérios sejam atendidos, pois serão usados nas análises das atividades e programas posteriores.

3.4 PSP1.1 PLANEJAMENTO DE TEMPO

O Planejamento de tempo é uma tarefa importante a ser considerada pelo engenheiro de software. Para que seja possível assumir compromissos que possam ser cumpridos, há a necessidade de identificar quanto tempo uma tarefa precisa para ser executada. Os compromissos assumidos anteriormente, e que ainda não estão concluídos precisam ser levados em conta na hora de assumir novos trabalhos.

O engenheiro de software no planejamento das atividades diárias, precisa decidir como quer gastar o seu tempo. A primeira decisão é estipular o número de horas de trabalho durante o dia. Após isto, é necessário ser feito a análise das tarefas que precisam ser realizadas, e posteriormente, definir o número de horas por dia gastos em cada uma das tarefas. Com estes dados em mãos, é possível elaborar o cronograma com a hora de início e fim de cada atividade.

À medida em que o engenheiro de software tenha dados sobre quanto tempo é necessário para completar cada uma das tarefas, o desenvolvimento do cronograma de atividades, torna-se uma tarefa possível de ser executada. Porém, um problema na engenharia de software, é conseguir estipular o tempo que será gasto no desenvolvimento de um programa ou de outras atividades, como planejamento, compilação etc.

O PSP1.1 apresenta ao engenheiro de software como elaborar o planejamento e a estimativa de tempo necessário para o desenvolvimento de um produto de software. A estimativa de tempo provida pelo PSP1.1 quando combinado com o método de estimativa PROBE e com dados históricos de custo e tamanho de cinco ou seis programas, torna o engenheiro de software apto a fazer bons planos de desenvolvimento (HUMPHREY, 1995).

O objetivo do PSP1.1 é introduzir e praticar métodos para que o engenheiro de software elabore o plano de tempo e horários, monitore o desempenho pessoal em relação a estes planos e que julgue a data provável de término do projeto. Para a

elaboração de um cronograma de atividades é necessário ter em mãos um plano detalhado do tempo disponível de cada um dos envolvidos no projeto.

3.4.1 Estimando o tempo

Com a utilização de processo de desenvolvimento de software estruturado e de um conjunto de dados históricos de projetos anteriores, é possível produzir planos com compromissos que podem ser cumpridos. À medida em que ganha experiência e habilidade com os planos, o engenheiro de software adquire confiança em suas estimativas, e conseqüentemente, convicção para argumentar os prazos e custos expostos no plano. Segundo Humphrey, (1995) a habilidade no planejamento pode proporcionar o melhor relacionamento com sua equipe de trabalho e com seu gerente, e o torna um engenheiro de software mais seguro e confiante.

O processo de planejamento do trabalho inicia com a estimativa de tamanho do produto a ser desenvolvido. Após isto, é estimado o tempo necessário para execução da tarefa, e por fim produzido o cronograma de atividades. Esta atividade é feita relacionando as horas, de desenvolvimento, gastas em projetos anteriores com as estimativas atuais de tamanho.

Para que o engenheiro de software possa elaborar um projeto, é necessário o plano detalhado dos recursos necessários. É feita a estimativa de quantas horas diretas espera-se gastar em cada tarefa. Também é necessário fazer a estimativa do tempo total disponível que será gasto na elaboração do produto. Esta projeção deve permitir outros compromissos nas atividades diárias do engenheiro de software. Alguns engenheiros acreditam que gastam mais ou menos 50% do seu tempo no trabalho direto no projeto (HUMPHREY, 1995).

Com a estimativa do tempo de cada tarefa e um plano de recursos detalhado é possível calcular quando cada tarefa se inicia e termina. Com estes dados o engenheiro consegue planejar as datas para os pontos chaves do projeto, como por exemplo o início e término das fases de desenvolvimento. Estas datas são usadas como fundamentação sólida para assumir compromissos. A qualidade da estimativa do tempo de desenvolvimento depende da quantidade e qualidade dos dados históricos que forem registrados.

3.4.2 Conteúdo do PSP1.1

Assim como os níveis anteriores do PSP, o PSP1.1 possui um processo estruturado de desenvolvimento que precisa ser seguido pelo engenheiro de software.

No desenvolvimento do PSP1.1, são utilizados os seguintes scripts, formulários, modelos e padrões, que estão inclusos no Anexo A, de acordo com a indicação.

Script de Processo PSP1.1	**Quadro A.33**
Script de Planejamento PSP1.1	**Quadro A.34**
Script de Desenvolvimento PSP1.1	**Quadro A.35**
Script de PostmortemPSP1.1	**Quadro A.36**
Resumo de Plano de Projeto PSP1 e Instruções	**Quadro A.37 e A.38**
Modelo de Planejamento de Tarefas e Instruções	**Quadro A.39 e A.40**
Modelo de Planejamento de Horário e Instruções	**Quadro A.41 e A.42**
Script de Estimativa PROBE	Quadro A.26
Modelo de Relatório de Teste e Instruções	Quadro A.29 e A.30
Modelo de Estimativa de Tamanho e Instruções	Quadro A.31 e A.32
PIP Proposta de Melhoria do Processo e Instruções	Quadro A.18 e A.19
Padrão de código	Quadro A.81
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

3.4.3 Novos elementos do processo

Como incremento em relação ao PSP1, dois novos elementos de processo são incluídos. Modelo de Planejamento de Tarefa e Modelo de Planejamento de Cronograma, Quadro A.39 e Quadro A.41, respectivamente. Estes modelos são tipicamente usados em tarefas que possuem duração de vários dias ou semanas.

3.4.4 Avaliação dos dados do PSP1.1

Para o aproveitamento da melhor maneira possível dos dados providos pelo PSP, os seguintes itens devem ser observados:

a) os dados do processo estão completos;
b) os dados são precisos e auto-consistentes;
c) o relatório do processo é submetido no formato e ordem própria;
d) os dados históricos são usados no planejamento do trabalho;

Para que seja possível o acompanhamento da performance do engenheiro de software, há a necessidade da avaliação constante do desempenho e das medidas providas pelo engenheiro de software através dos formulários. O PSP provê os dados que devem ser avaliados em cada nível.

3.5 MELHORANDO A QUALIDADE DO SOFTWARE

No desenvolvimento de software, é possível ver casos onde sejam gastos 40% do esforço do projeto total na fase de testes do sistema (PRESSMANN, 1995). Mas por que é gasto tempo em testes de software? Um problema é a falta de qualidade durante o processo de desenvolvimento do produto, onde a não utilização de um processo definido pode ser visto como uma das causas principais. Este percentual alto também decorre do esforço na tentativa de entregar o produto com a menor quantidade de erros possível.

Porém, em alguns casos os testes são ineficientes, restando erros no software. Uma maneira eficiente para encontrar erros em um produto é através da revisão de projeto e código. Com as revisões, é possível encontrar os defeitos, diretamente, enquanto que nos testes obtém-se somente os sintomas. A grande diferença entre estas duas abordagens é chamada depuração. Ao fazer a revisão de um programa, o engenheiro de software sabe onde está e o que a lógica do programa está fazendo, tornando assim as correções mais completas e corretas (JUNIOR, 2000).

No PSP2 é mostrado ao engenheiro de software a importância das revisões de projeto e código. Um dos objetivos da realização de revisões, de projeto e código, é melhorar a qualidade e produtividade do trabalho do engenheiro de software. As revisões de código e projeto apresentam melhoras significativas, em relação a qualquer

outra mudança feita no processo pessoal de desenvolvimento de software. (HUMPHREY, 1995).

Em sistemas maiores as revisões devem ser feitas nos requisitos, especificações e código, isto possibilita a diminuição do número de erros no sistema. Uma sugestão para a revisão de código, é que esta seja feita antes da primeira compilação do programa, para que seja possível encontrar e analisar cada um dos erros do programa fonte. Ao fazer a revisão o engenheiro de software deve corrigir todos os problemas encontrados de lógica, estrutura e legibilidade do programa. (HUMPHREY, 1995).

Há situações onde o programa pode ter sido escrito de maneira obscura ou confusa. Nesta situação, algumas vezes pode ser mais fácil reescrevê-lo do que corrigí-lo. Os programas devem ser escritos de maneira fácil de ler e entender, podendo ser feitos comentários no programa, para ajudar a compreender mais facilmente o programa. Os programas precisam ser escritos de maneira que o programador possa mostrar a outro programador sem ficar constrangido (HUMPHREY, 1995).

Alguns engenheiros de software acreditam que parte dos erros cometidos no desenvolvimento do software seja "erros tolos". Esta situação faz com que ele acredite que estes erros possam ser resolvidos pelo processo de tentativa e erro (escreve, compila, corrige, escreve), isto pode dificultar a situação.

Há necessidade de prevenir os defeitos, e pode ser feito através do gerenciamento de defeitos. O engenheiro de software deve registrar e analisar onde os erros são injetados e removidos do sistema. Analisando este registro, o engenheiro de software consegue saber quais os erros cometidos, e então pode construir uma lista de verificação (*checklist*) com os principais erros cometidos, para depois fazer revisão de projeto e código.

3.5.1 O que são as revisões

O objetivo da revisão é auxiliar o engenheiro de software a tratar objetivamente os defeitos inseridos no projeto e no código, removendo-os antes da primeira compilação. As revisões devem ser utilizadas como um "filtro" para o processo de desenvolvimento de software, retirando e registrando todos os defeitos encontrados. As revisões podem ser aplicadas em vários pontos do processo de desenvolvimento (levantamento dos

requisitos, projeto, codificação), e servem para detectar defeitos que possam ser eliminados, de preferência nas fases iniciais do projeto onde o custo geralmente, é menor.

Freedman e Weinberg, *apud* Pressmann (1995, p.736), discutem a necessidade de fazer revisões da seguinte maneira:

> O trabalho técnico precisa de revisão pelo mesmo motivo que os lápis precisam de borrachas: Errar é humano. A segunda razão pela qual precisamos de revisões técnicas é que, não obstante as pessoas captem bem alguns de seus próprios erros, grandes classes de erros escapam mais facilmente a quem lhes deu origem do que a outras pessoas. O processo de revisão é, por conseguinte, a resposta à oração de Robert Burns:
> Oh, quem nos dera o poder de poder nos vermos a nós mesmos da mesma forma que os outros no vêem.
> Uma revisão – qualquer revisão – é uma maneira de usar a diversidade de um grupo de pessoas para:
> 1. Apontar melhorias necessárias ao produto de uma única pessoa ou equipe.
> 2. Confirmar as partes de um produto em que uma melhoria não é desejada ou não é necessária.
> 3. Realizar um trabalho técnico com uma qualidade mais uniforme ou, pelo menos, mais previsível do que aquilo que pode ser realizado sem revisões, de forma a tornar esse trabalho técnico mais administrável.

A revisão de projeto e código é uma tarefa importante a ser realizada durante o desenvolvimento de código. Quando estas são realizadas com atenção e seguindo a lista de verificação (*checklist*) de projeto e código, podem ter importância na remoção de defeitos do sistema e conseqüente aumento da qualidade do produto.

Os princípios básicos das revisões são os seguintes (HUMPHREY, 1995):

1. estabelecer as metas da revisão. No PSP a meta das revisões é encontrar e corrigir todos os erros antes da primeira compilação. No início do processo de

revisão consegue-se encontrar alguns erros, porém outros ainda são perdidos. É importante registrar os dados das revisões, e que depois sejam analisados para que sejam feitas as alterações e melhorias do processo de revisão;

2. seguir um processo definido para a revisão. O processo de revisão é muito parecido com outros processos. Requer critérios de entrada, um conjunto de tarefas e critérios de saída. A lista de verificação da revisão de projeto é mostrada no Quadro A.49 e a lista de verificação da revisão de código é mostrada no Quadro A.50;
3. medir e melhorar o processo de revisão. As medidas das revisões são utilizadas para melhorar a qualidade das revisões. Geralmente, uma revisão de alta qualidade é aquela que encontra o maior número de defeitos no menor espaço de tempo. Para avaliar a qualidade da revisão, deve-se medir o tempo gasto nas revisões e registrar todos os defeitos encontrados, bem como é preciso registrar os defeitos encontrados mais tarde. Estes dados devem ser usados para melhorar a qualidade e eficiência das revisões.

As revisões de projeto e código são parte importante do desenvolvimento de software. Assim como o processo de desenvolvimento, as revisões também devem seguir um processo estruturado para seu desenvolvimento.

3.5.2 Revisão de projeto

A revisão de projeto não é um processo simples de ser feito. Existem alguns itens que devem ser analisados e seguidos para que o engenheiro de software possa fazer revisões. Um dos requisitos principais para a realização de revisão de projeto é que esteja sendo utilizado um processo definido e estruturado de desenvolvimento de software. Em um processo de desenvolvimento que não utilizam um processo definido para desenvolvimento, o engenheiro pode por exemplo, estar fazendo projeto e codificação como uma fase única. Esta situação dificulta a revisão, o que deve ser revisado? O projeto ou a codificação?

Para que as revisões possam ser efetuadas há a necessidade de produzir projetos que possam ser revisados. Para um projeto poder ser revisado, seus objetivos e funções

devem estar explícitos. Ao fazer a revisão de um projeto, o engenheiro de software deve ter a lista com as funções que devem ser atendidas, as regras e condições que devem ser satisfeitas.

As organizações de desenvolvimento de software que utilizam um processo definido e estruturado de desenvolvimento, possuem um padrão de projeto que é utilizado em todos os projetos. Também podem existir padrões particulares de um software ou sistema. Sem a utilização destes padrões, a base para desenvolvimento da revisão será de baixa qualidade. Isto significa que o produto revisado poderá ter problemas não detectados pela revisão. Os padrões definidos são a condição essencial para projetos revisáveis (HUMPHREY, 1995).

O engenheiro de software deve seguir uma estratégia explícita de revisão. Esta estratégia, estabelece a ordem na qual são examinados elementos do projeto, como lógica, casos especiais, padrões etc. Esta ordem depende da estrutura do projeto que está sendo revisado e revisão deve ser parte da estratégia de desenvolvimento.

A revisão do projeto pode ser executada em estágios. Para a revisão de cada segmento do projeto, considere as seguintes condições: (1) Verificar se todos os elementos requisitados do programa foram desenvolvidos; (2) Verificar a estrutura e fluxo global dos programas; (3) Verificar se a lógica de construção do programa está correta; (4) Verificar a robustez dos programas; (5) Verificar todas as chamadas de métodos, funções e procedimentos para garantir que todos os parâmetros e tipos são especificados e que cada função é usada adequadamente; (6) Verificar todas as variáveis especiais, parâmetros, tipos de dados e arquivos (HUMPHREY, 1995).

O engenheiro de software deve verificar se a lógica implementa corretamente os requisitos. Esta verificação pode parecer simples de fazer, porém pode envolver um esforço grande de trabalho. Este trabalho envolve revisar todos os requisitos e verificar se todos eles são cobertos pelo projeto.

3.5.3 Revisão de código

Algumas pessoas acreditam que podem escrever artigos na primeira tentativa. Porém, escritores profissionais sabem que o segredo de uma boa escrita é reescrever. Escrever programas é a mesma coisa. Alguns programadores pensam que podem

escrever programas de alta qualidade, isto é, programas estruturados dentro do padrão de codificação, na primeira tentativa. Porém, desta maneira eles podem ter problemas para compilar e testar este programa, pois é programa propenso a ter defeitos (HUMPHREY, 1995).

A revisão de código é uma maneira de encontrar defeitos. A revisão de código é a análise do código fonte na procura de erros de sintaxe, digitação, lógica etc. Esta revisão deve ser feita antes da primeira compilação e teste. Alguns dos defeitos de softwares são resultado de omissões simples e ingênuas do engenheiro de software, e podem ser encontradas após a produção do projeto ou do código. A maneira tradicional de fazer a revisão de código é utilizando uma listagem impressa do código fonte, revisando linha por linha. As listagens permitem mover-se entre os segmentos de código, fazer anotações e verificar as seções completadas.

Embora as revisões de código sejam consumidoras de tempo, elas são mais eficientes que os testes. Dados de estudantes e engenheiros de software mostram que as revisões de código são entre três e cinco vezes mais eficientes que as execuções iniciais dos testes. Por exemplo, um engenheiro encontra de 2 a 4 erros por hora em uma unidade de teste, enquanto encontra de 6 a 10 erros por hora na revisão de código. A explicação para as revisões serem mais eficientes é que na revisão o engenheiro trabalha diretamente com o problema e não com os sintomas (HUMPHREY, 1997).

As revisões têm desvantagens. Elas são consumidoras de tempo e difíceis de serem executadas corretamente. Porém, elas podem ser aprendidas e melhoradas com a prática no cotidiano.

3.5.4 O custo dos defeitos de software

Um dos grandes benefícios das revisões é a descoberta de defeitos nas fases iniciais do sistema, de maneira que cada defeito possa ser corrigido antes do passo seguinte do processo de desenvolvimento de software. Com a detecção e correção dos erros precocemente, o processo de revisão reduz substancialmente o custo dos passos subseqüentes do processo e manutenção do software (PRESSMANN, 1995).

Um erro descoberto durante a fase de projeto do sistema custa 1,0 unidade monetária para ser corrigido. Este mesmo erro, sendo descoberto logo antes de iniciar as

atividades de teste, custará 6,5 unidades, e durante o teste, 15 unidades. Porém, após o término do software, entre 60 e 100 unidades (PRESSMANN, 1995). Esta é uma das principais razões para a remoção de defeitos nas fases iniciais do projeto, onde o custo é menor.

As duas principais razões para que os defeitos sejam removidos o mais cedo possível, é o custo menor para correção e produzir um software com menos defeitos. Um programa que foi escrito com problemas, este é assim para sempre. Ele pode ser corrigido várias vezes, porém sempre é defeituoso e precisa de correções durante sua utilização (HUMPHREY, 1995).

Os engenheiros que toleram o processo de desenvolvimento de software não estruturado, provavelmente, irão produzir produtos de baixa qualidade. Se um engenheiro de software diz que está muito ocupado para fazer uma correção, e que irá fazer isto mais tarde, provavelmente jamais produzirá software de alta qualidade. Para produzir software de qualidade, cada passo de desenvolvimento deve ser de qualidade. Enquanto que práticas rigorosas de qualidade podem parecer caras e onerosas, elas normalmente levam o mesmo tempo para serem feitas (HUMPHREY, 1995).

3.5.5 Usando a revisão para encontrar defeitos

O objetivo da revisão de código é encontrar os defeitos o mais cedo possível no processo de software. É também encontrar estes defeitos no menor tempo possível. É importante seguir algumas práticas durante a revisão de código, como as seguintes:

a) fazer a revisão antes da primeira compilação;
b) fazer a revisão em uma listagem impressa do código fonte;
c) registrar todos os defeitos encontrados no log de registro de defeito;
d) durante a revisão, registrar e corrigir todos os tipos de defeitos encontrados.

As revisões devem ser usadas para melhorar a qualidade dos produtos, através da remoção de defeitos o mais cedo possível. Para que o engenheiro de software consiga desenvolver revisões de qualidade, um item que deve ser considerado é a utilização da lista de verificação (*checklist*) com todos os itens que devem ser verificados.

3.5.6 Listas de verificação

Uma maneira para ter sucesso na revisão, ou seja, encontrar e corrigir o maior número de erros possíveis, é utilizar um procedimento eficiente. Uma lista de verificação contém o conjunto de passos que devem ser seguidos precisamente durante a revisão. Normalmente, quando as pessoas têm coisas importantes para fazer, pode ser benéfico fazer exatamente como foi especificado seguindo uma lista de verificação (HUMPHREY, 1995).

A lista de verificação deve conter todos os itens e procedimentos que devem ser avaliados na execução da revisão de projeto ou código. A não utilização da lista pelo engenheiro de software pode deixar passar despercebido, itens importantes do projeto ou do código com erros. Quando há a necessidade de encontrar e corrigir todos os defeitos em um programa, o engenheiro de software deve fazer uso de um procedimento preciso. A lista de verificação auxilia o engenheiro a seguir os procedimentos. As Listas de Verificação de Revisão de Código e Projeto com alguns itens a serem observados são mostrados nos Quadro A.49 e Quadro A.50, respectivamente.

Para que estas listas tenham a devida utilidade, devem ser analisadas antes do início de qualquer projeto novo. Se houver a necessidade de alterações ou inclusões de novos itens de verificação, devem ser incluídos e seguidos no novo projeto.

3.5.7 Conteúdo do PSP2

Assim como os níveis anteriores do PSP, o PSP2 possui um processo estruturado de desenvolvimento que precisa ser seguido pelo engenheiro de software.

No desenvolvimento do PSP2, são utilizados os seguintes scripts, formulários, modelos e padrões, que estão inclusos no Anexo A, de acordo com a indicação.

***Script* de Processo PSP2**	**Quadro A.43**
***Script* de Planejamento PSP2**	**Quadro A.44**
***Script* de Desenvolvimento PSP2**	**Quadro A.45**
Script* de *Postmortem PSP2	**Quadro A.46**
Resumo de Plano de Projeto PSP2 e Instruções	**Quadro A.47 e A.48**
Lista de Verificação para Revisão de Projeto	**Quadro A.49**

Lista de Verificação para Revisão de Código	**Quadro A.50**
Modelo de Planejamento de Tarefas e Instruções	Quadro A.39 e A.40
Modelo de Planejamento de Horário e Instruções	Quadro A.41 e A.42
Script de Estimativa PROBE	Quadro A.26
Modelo de Relatório de Teste e Instruções	Quadro A.29 e A.30
Modelo de Estimativa de Tamanho e Instruções	Quadro A.31 e A.32
PIP Proposta de Melhoria do Processo e Instruções	Quadro A.18 e A.19
Padrão de código	Quadro A.81
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

3.5.8 Avaliação dos dados do PSP2

Para o aproveitamento da melhor maneira possível dos dados providos pelo PSP, os seguintes itens devem ser observados:

- os dados do processo estão completos.
- os dados são precisos e auto-consistentes.
- o relatório do processo é submetido no formato e ordem própria.
- Os dados históricos são usados no planejamento do trabalho.

Um quinto critério de avaliação do processo é exigido no PSP2. Que os dados históricos sejam usados na melhoria do processo, como por exemplo, fazer a atualização das listas de revisão de projeto e código. Para que este critério seja atendido, deve-se examinar os dados dos mais recentes, antes de cada novo projeto, para detectar se alguma lista de verificação precisa ser modificada. Todas as mudanças necessárias devem ser efetivadas, e estas alterações devem ser usadas pelo engenheiro de software nos próximos projetos. Estas alterações também precisam ser registradas no formulário PIP.

É importante que todos estes critérios sejam atendidos, pois serão usados nas análises das atividades e programas posteriores.

3.6 PSP2.1 – PROJETO DE SOFTWARE

O PSP2.1 apresenta ao engenheiro de software quatro modelos de projeto (Modelo de Cenário Operacional, Modelo de Especificação Funcional, Modelo de Especificação de Estado, Modelo de Especificação Lógica). Estes modelos provêem a estrutura para o registro de pontos fundamentais dos projetos. Embora os modelos não mostrem ao engenheiro de software como fazer o projeto, podem ajudar no registro dos dados do projeto.

O objetivo do PSP2.1 é ajudar a reduzir o número de defeitos nos projetos, provendo critérios para identificar se um projeto está completo, além de prover a estrutura para verificar a qualidade dos projetos. Com a utilização dos modelos propostos pelo PSP2.1 o engenheiro de software pode fazer o registro e acompanhamento dos itens que compõem o projeto de software.

Antes que um projeto de software possa ser planejado, é necessário estabelecer os objetivos e o escopo do projeto. Soluções alternativas devem ser observadas e as restrições administrativas e técnicas identificadas. A falta dessas informações inviabiliza a definição das estimativas de custo, a divisão das tarefas ou a programação de um projeto administrável (PRESSMAN 1995).

O objetivo do projeto de software é transformar os requisitos definidos na especificação do sistema em um produto implementável. A qualidade deste projeto é importante, pois é quase impossível implementar um produto de alta qualidade, com um projeto de qualidade duvidosa. Um projeto de qualidade tem duas partes que devem ser observadas: a qualidade do conteúdo do projeto e a qualidade da representação do projeto. Um projeto de alta qualidade pode ser implementado com problemas, se a sua representação tiver problemas (PRESSMAN 1995).

O projeto de software é um processo criativo, e não pode ser reduzido a um procedimento rotineiro. Um projeto normalmente inicia com a definição dos objetivos do produto, juntando os dados relevantes, produzindo uma avaliação do projeto e especificando todos os detalhes. Estes passos não são tarefas seqüenciais isoladas, elas são cooperativas e atividades paralelas. Um projeto envolve descobertas e invenções e, freqüentemente precisa de saltos intuitivos de um nível de abstração para outro (PRESSMANN, 1995).

O projeto de software é uma fase muito importante no desenvolvimento de software. É a partir do projeto que é feita a implementação dos dados do sistema em uma linguagem de programação. Portanto, se o projeto tiver problemas, estes problemas serão repassados à implementação. O acompanhamento e registro de todos os detalhes do projeto de software, são úteis para que seja possível garantir que o projeto atende todos os requisitos do sistema.

3.6.1 O projeto conceitual

Ao iniciar um projeto de software, geralmente os requisitos iniciais estão incompletos e imprecisos. O primeiro passo de um projeto é obter todas as informações possíveis do produto a ser desenvolvido. Conseguir estas informações, em programas pequenos pode até ser simples, porém em programas maiores a complexidade aumenta substancialmente. A complexidade dos requisitos é determinada pela complexidade do problema que está sendo resolvido. Os problemas podem ter soluções complexas, porém, isto nem sempre é verdade. A principal exceção é quando os elementos do problema podem ser identificados e resolvidos antes. Neste caso, podem ser reutilizadas partes das soluções prévias (HUMPHREY, 1995).

Similarmente, problemas simples, nem sempre tem soluções simples. Em algumas situações, requisitos simples podem causar problemas, pois a simplicidade dos requisitos pode ser enganosa. Aplicações que parecem simples para algumas pessoas, podem ser difíceis de implementar.

Outro problema que pode ocorrer é o grande volume de informações. Por um lado, é preciso assegurar-se que sejam captadas todas as informações críticas necessárias. Por outro lado, é necessário evitar o grande número de detalhes. Para evitar problemas é útil seguir um processo interativo (HUMPHREY, 1995):

1. manter o foco em assuntos de alto nível até que se tenha domínio para produzir todo projeto conceitual;
2. completar e documentar o projeto conceitual;
3. fazer e documentar o plano de desenvolvimento;
4. uma vez que se tenha o projeto conceitual, seus dados provêem as mudanças do processo. Há a necessidade de testar o projeto conceitual,

buscando as informações para validar o modelo ou pelo menos definir o escopo e limites. O engenheiro de software precisa tomar cuidado para não se perder em detalhes, é preciso cercar o problema olhando este de todos os lados que se possa imaginar. É necessário pensar como um usuário, visualizando todos os cenários possíveis. Então, é preciso caminhar por estes cenários com o projeto conceitual e assegurar que estes são controlados pelo projeto. Se houver muitas exceções, ou se o seu projeto conceitual não estiver se ajustando naturalmente ao problema, é necessário revisá-lo;

5. assim que o projeto conceitual estiver sólido, pode-se dirigir o foco aos detalhes. O projeto conceitual agora provê uma estrutura para catalogar os conhecimentos adquiridos e assegurar que não se perca em detalhes.

3.6.2 O Modelo de Especificação de Estado

O Modelo de Especificação de Estado descreve o estado de comportamento do objeto e contém uma descrição de cada estado do objeto e a transição entre eles. Este modelo contém o nome do estado e o resumo ou descrição matemática do estado, junto com os valores do atributo para o estado. Depois são descritos os próximos estados, explicando as condições de transição entre eles. Um Modelo de Especificação de Estado é mostrado no Quadro A.62. Quando o engenheiro de software não está fazendo uso de métodos orientados a objetos, pode ser utilizado para descrever funções ou procedimentos.

É fácil assumir que uma transição de estado é possível, após ter descoberto que seja possível debaixo de algumas condições incomuns. Estas condições incomuns são freqüentemente a fonte de defeitos de sistema, assim, devem ser observadas durante o projeto as razões para cada transição impossível (HUMPHREY, 1995).

O engenheiro de software deve ter cuidado para descrever todas as condições de transições precisamente. A estrutura de estado muito complexa pode ser um sintoma de um projeto excessivamente complicado que pode causar problemas de implementação, teste e manutenção. Todas as situações que podem causar problemas nas fases

posteriores do processo de desenvolvimento, devem ser observadas e resolvidas com a maior brevidade.

Um programa razoavelmente complexo pode ter muitos estados, logo, é necessário identificar aqueles que têm uma significação lógica. Uma diretriz útil, é que um estado é caracterizado por valores de atributo diferentes e comportamento de objeto diferente. Reciprocamente, se o comportamento do objeto for idêntico em dois estados diferentes, esses estados deveriam ser fundidos, embora possam ocorrer diferenças de valores de parâmetros (HUMPHREY, 1995).

3.6.3 O Modelo de Especificação Lógica

O Modelo de Especificação Lógica possui a lógica do código fonte para cada objeto ou elemento do programa. Em casos onde o engenheiro de software não está fazendo uso de métodos orientados a objetos, pode ser utilizado para a especificação de funções, procedimentos ou a lógica da rotina principal.

Este modelo é quase um programa fonte completo. Os detalhes das instruções, algumas variáveis, declarações de parâmetros e inicializações permanecem para serem completadas durante a implementação. O número de referência lógica é usado para facilitar as revisões. Para prover o máximo de ajuda, este número deve identificar cada ponto de ação e decisão lógica no pseudocódigo. Um Modelo de Especificação Lógica é mostrado no Quadro A.64.

3.6.4 O Modelo de Especificação funcional

O Modelo de Especificação Funcional descreve as especificações funcionais dos métodos providos por um objeto. O nome da classe ou objeto é listado no topo do modelo junto com as classes das quais são descendentes diretos e com os atributos do objeto. Depois são listados os métodos do objeto, como a declaração do método é incluída no programa fonte, seguido por uma declaração precisa do que o método provê. O objetivo é mostrar no menor número de linhas possíveis, o que exatamente o método faz. Um Modelo de Especificação Funcional é mostrado no Quadro A.60. Este modelo

pode ser usado para descrever funções ou procedimentos em métodos não orientados a objeto.

3.6.5 O Modelo de Cenário Operacional

O Modelo de Cenário Operacional descreve o comportamento operacional do sistema em um ou mais cenários. O objetivo é ajudar a visualizar como o programa reage sob vários cenários típicos de usuário. Quando cara a cara com uma decisão de projeto que envolve uma ação do usuário, deve ser produzido um cenário para tentar ver como reage com o usuário. Este modelo possui as descrições dos cenários operacionais que serão seguidos na utilização do programa e é usado para garantir que o engenheiro considere o uso de todos os casos. Também pode ser usado para especificar cenários de teste.

Este cenário ajuda a visualizar o comportamento e determinar a escolha apropriada do projeto. Enquanto nenhum conjunto de cenários pode cobrir todas as combinações de uso para cada sistema moderadamente complexo, deve-se incluir exemplos de funções chaves e condições de exceção.

Com um planejamento cuidadoso, porém, pode ser atendido um grande percentual de casos importantes com poucos cenários. Se não forem construídos e testados os cenários, provavelmente algumas ações chaves são negligenciadas. Freqüentemente, estas ações negligenciadas não são nem mesmo testadas e podem ser uma fonte significante de aborrecimento do usuário. Um Modelo de Cenário Operacional é mostrado no Quadro A.79.

3.6.6 Conteúdo do PSP2.1

Assim como os níveis anteriores do PSP, o PSP2.1 também possui um processo estruturado de desenvolvimento que precisa ser seguido pelo engenheiro de software.

No desenvolvimento do PSP2.1, são utilizados os seguintes scripts, formulários, modelos e padrões, que estão inclusos no Anexo A, de acordo com a indicação.

***Script* de Processo PSP2.1** **Quadro A.51**

***Script* de Planejamento PSP2.1**	**Quadro A.52**
***Script* de Desenvolvimento PSP2.1**	**Quadro A.53**
***Script* de *Postmortem* PSP2.1**	**Quadro A.54**
Resumo de Plano de Projeto PSP2 e Instruções	**Quadro A.55 e A.56**
Modelo de Cenário Operacional e Instruções	**Quadro A.58 e A.59**
Modelo de Especificação Funcional e Instruções	**Quadro A.60 e A.61**
Modelo de Especificação de Estado e Instruções	**Quadro A.62 e A.63**
Modelo de Especificação Lógica e Instruções	**Quadro A.64 e A.65**
Lista de Conferência de Revisão de Projeto do PSP2.1	**Quadro A.57**
Lista de Verificação para Revisão de Código	Quadro A.50
Modelo de Planejamento de Tarefas e Instruções	**Quadro A.39 e A.40**
Modelo de Planejamento de Horário e Instruções	**Quadro A.41 e A.42**
Script de Estimativa PROBE	Quadro A.26
Modelo de Relatório de Teste e Instruções	Quadro A.29 e A.30
Modelo de Estimativa de Tamanho e Instruções	Quadro A.31 e A.32
PIP Proposta de Melhoria do Processo e Instruções	Quadro A.18 e A.19
Padrão de código	Quadro A.81
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

3.6.7 Avaliação dos dados do PSP2.1

Os critérios para avaliação dos dados do PSP2.1 são os mesmos do PSP2, como segue.

a) os dados do processo estão completos;
b) os dados são precisos e auto-consistentes;
c) o relatório do processo é submetido no formato e ordem própria;
d) os dados históricos são usados no planejamento do trabalho;
e) são constantemente usados dados históricos para melhorar o processo.

3.7 O PROCESSO CÍCLICO

O tamanho de produtos de software aumentou significativamente nos últimos anos. Enquanto a taxa de aumento varia entre os tipos de produto, o tamanho tem aumentado em uma ordem de 5 a 10 vezes. Isto ocorre, inclusive nas situações onde o mesmo produto é implementado, porém com uma nova tecnologia. Um exemplo é a impressora laser da Hewlett-Packard (HP). O produto inicial usava 25.000 LOC, o modelo seguinte usava 200.000 LOC, enquanto que o terceiro modelo usava 1.000.000 LOC. Constata-se um aumento de 40 vezes o tamanho inicial, num período de aproximadamente 10 anos (HUMPHREY, 1995).

A cada ano que passa, os clientes demandam por produtos mais sofisticados. Isto faz com que o software de controle destes produtos, também sejam mais sofisticados, exigindo mais código e um processo de larga escala. O tamanho dos próximos produtos desenvolvidos, tende a ser maior que os produtos desenvolvidos hoje. Embora o engenheiro de software continue a fazer o desenvolvimento de componentes de software, seu trabalho envolve grandes equipes ou até mesmo, grupos de equipes. Um processo ótimo para um projeto, talvez não o seja para um projeto 5 ou 10 vezes maior. Como o tamanho do produto de software que é desenvolvido aumenta, seu processo deve aumentar também (HUMPHREY, 1995).

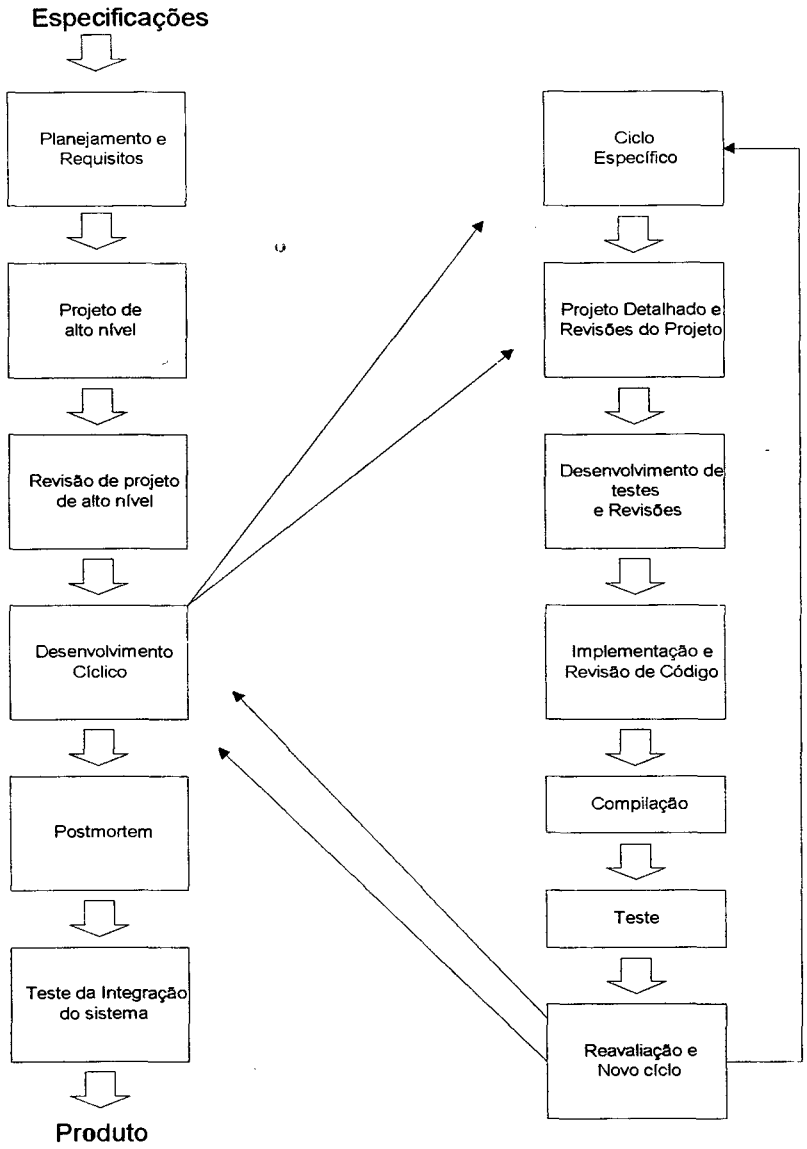
Como o aumento de tamanho dos produtos de software, há a necessidade de planejar o desenvolvimento de grandes produtos. Como desenvolver grandes produtos de software? O PSP3 provê um método de desenvolvimento cíclico, onde grandes programas são divididos em programas menores, variando entre 100 e 300 LOCs.

3.7.1 O Desenvolvimento Cíclico

O PSP3 apresenta o desenvolvimento cíclico projetado para auxiliar o engenheiro de software a desenvolver grandes programas. O objetivo do PSP3 é aumentar a capacidade do processo pessoal para desenvolver programas de milhares de linhas de código. O PSP3 é uma fundamentação para o processo pessoal no desenvolvimento de software em larga escala. Conseqüentemente, o engenheiro de software deve ser capaz

de controlar produtos de grande complexidade e relacionar o processo de desenvolvimento para sua equipe. O uso do processo cíclico do PSP3 é mostrado na figura 3.4.

Figura 3.1 O Processo do PSP3



Fonte: HUMPHREY, 1995

3.7.2 Planejamento e Requisitos

O PSP3 inicia da fase de planejamento e levantamento dos requisitos. Aqui é produzido o projeto conceitual de todo sistema, estimativa de tamanho e planos para o desenvolvimento do trabalho. Nesta fase, também são definidos o escopo e os limites do sistema. Há a necessidade de fazer o planejamento e levantamento dos requisitos, com precisão, pois, os erros que não forem encontrados aqui, irão causar problemas nas fases posteriores.

3.7.3 Projeto de Alto Nível

No projeto de alto nível são identificadas as divisões naturais do produto, e produzida uma estratégia cíclica. Uma regra utilizada, é que cada ciclo esteja entre 100 e 300 LOC entre novas e modificadas. Nas primeiras divisões do produto, alguns desses ciclos ficam maiores do que o planejado, porém é necessário ir fazendo as melhorias na divisão para que se tenham ciclos pequenos, pois são mais fáceis de entender e trabalhar. A combinação de várias funções em um mesmo ciclo, somente pode ser feita se estas funções forem simples e bem entendidas pelo engenheiro de software. Depois de pronto o projeto de alto nível, é feita a revisão e, após isto ocorre o desenvolvimento cíclico.

3.7.4 Desenvolvimento Cíclico

O desenvolvimento cíclico inicia com a especificação do ciclo corrente. Estas especificações devem ser desenvolvidas durante o projeto de alto nível, porém elas não são necessárias antes do ciclo de desenvolvimento. Cada ciclo é essencialmente um processo PSP2.1 que produz uma parte do produto. O engenheiro de software deve observar que as fases de planejamento e postmortem são feitas uma única vez para o projeto completo.

Durante os ciclos, as revisões e testes devem ser tão completas quanto possível. Cada ciclo é a fundamentação para o ciclo seguinte. Assim, qualquer defeito em um dos ciclos, pode causar problemas mais tarde.

O processo cíclico é efetivo pois constrói e testa as novas funções em uma fundamentação sólida. A personalização é preservada, desde que cada desenvolvimento incremental seja fechado e essencialmente livre de defeitos. Revisões de projeto completas e teste compreensivos são partes críticas do processo de desenvolvimento cíclico.

O engenheiro de software deve observar que o simples fato de dividir programas maiores em partes menores, não faz com que seus produtos sejam de qualidade. Há a necessidade de observar todos os detalhes e requisitos do sistema, fazendo revisões com precisão para que sejam encontrados e removidos todos os erros do projeto.

3.7.5 Teste de Integração do Sistema

O desenvolvimento cíclico inclui ainda um teste de desenvolvimento que pode mostrar problemas no projeto, como também problemas operacionais e de usuário. Outro assunto diz respeito à necessidade de dados de teste, modos especiais de teste, cenários de usuários e facilidade de suporte aos testes. A vantagem do planejamento e desenvolvimento dos testes no início, é que força o engenheiro de software a pensar sobre o produto a partir de uma perspectiva de teste.

Cada ciclo subseqüente adiciona funções e são progressivamente integrados ao produto previamente testado. Após o ciclo final, o engenheiro de software deve completar uma unidade combinada de teste de integração do programa inteiro.

3.7.6 Reavaliação e novo ciclo

Após cada ciclo, deve ser reavaliado o trabalho para determinar o seu estado e reavaliar o plano. Para grandes projetos, deve ser considerada a elaboração de relatórios de estado ao final de cada ciclo. Se os dados da avaliação indicarem desvios significativos do plano original, o plano deve ser ajustado e o gerente do projeto deve ser comunicado para que seja corrigido o plano de desenvolvimento.

3.7.7 Conteúdo do PSP3

Assim como os níveis anteriores do PSP, o PSP3 também possui um processo estruturado de desenvolvimento que precisa ser seguido pelo engenheiro de software.

No desenvolvimento do PSP3, são utilizados os seguintes scripts, formulários, modelos e padrões, que estão inclusos no Anexo A, de acordo com a indicação.Todos estes itens estão inclusos no Anexo A com as tabelas e números indicados.

Script de Processo PSP3	**Quadro A.66**
Script de Planejameto PSP3	**Quadro A.67**
Script de Projeto de Alto Nível do PSP3	**Quadro A.68**
Script de Revisão de Projeto de Alto Nível do PSP3	**Quadro A.69**
Script de Desenvolvimento PSP3	**Quadro A.70**
Script de Postmortem PSP3	**Quadro A.71**
Resumo de Plano de Projeto PSP3 e Instruções	**Quadro A.72 e A.73**
Resumo de Ciclo e Instruções	**Quadro A.74 e A.75**
Log de Acompanhamento de Assunto e Instruções	**Quadro A.77 e A.78**
Modelo de Cenário Operacional e Instruções	Quadro A.58 e A.59
Modelo de Especificação Funcional e Instruções	Quadro A.60 e A.61
Modelo de Especificação de Estado e Instruções	Quadro A.62 e A.63
Modelo de Especificação Lógica e Instruções	Quadro A.64 e A.65
Lista de Conferência de Revisão de Projeto do PSP3	**Quadro A.76**
Lista de Verificação para Revisão de Código	Quadro A.50
Modelo de Planejamento de Tarefas e Instruções	Quadro A.39 e A.40
Modelo de Planejamento de Horário e Instruções	Quadro A.41 e A.42
Script de Estimativa PROBE	Quadro A.26
Modelo de Relatório de Teste e Instruções	Quadro A.29 e A.30
Modelo de Estimativa de Tamanho e Instruções	Quadro A.31 e A.32
PIP Proposta de Melhoria do Processo e Instruções	Quadro A.18 e A.19
Padrão de código	Quadro A.81
Log de Registro de Tempo e Instruções	Quadro A.5 e A.6
Log de Registro de Defeito e Instruções	Quadro A.7 e A.8
Padrão de Tipo de Defeito	Quadro A.9

3.7.8 Novos elementos do processo PSP3

O Log de Acompanhamento de Assunto (ITL – *Issue Tracking Log*), provê um formulário conveniente para registrar propostas, problemas e questões abertas. Por exemplo: no meio de um projeto é decidido mudar o nome de uma variável para representar melhor seu objetivo. Ao invés de parar em cada ponto para verificar e assegurar que todas as ocorrências desta variável tenham sido alteradas, faz-se uma anotação no Log de Acompanhamento de Assunto e são observados posteriormente.

O Resumo de Ciclo (*The Cycle Summary*) é usado para registrar os dados do ciclo de desenvolvimento estimado e atual. Uma cópia do formulário de Resumo de Ciclo deve ser usada para registrar os planos do ciclo e outra para informar os resultados dos ciclos atuais. Estes dados podem ser facilmente obtidos ao final de cada ciclo, porém difíceis de serem reconstruídos posteriormente. Por exemplo: no registro de tamanho, o total de LOCs reutilizadas, adicionadas, apagadas ou modificadas podem ser contadas ao final de cada ciclo, porém isto é muito difícil de fazer algum tempo posterior.

3.7.9 Relatando especificações do processo

Os critérios para avaliação dos exercícios do PSP3 são os mesmos do PSP2.1, como segue.

a) Os dados do processo estão completos.
b) Os dados são precisos e auto-consistentes.
c) O relatório do processo é submetido no formato e ordem própria.
d) Os dados históricos são usados no planejamento do trabalho.
e) São constantemente usados dados históricos para melhorar o processo.
1. Um novo critério é incluído, que é a utilização dos dados e a experiência anterior, para que estes dados sejam usados para estabelecer metas de futuras melhorias no processo.

3.8 CONCLUSÃO

As medidas pessoais são importantes na utilização do PSP, pois provêem os dados sobre tempos e defeitos no processo de desenvolvimento. Estes dados devem ser utilizados para entender onde o engenheiro de software gasta seu tempo, bem como analisar onde são injetados e removidos os defeitos do software.

Com a utilização do PSP, é mostrado ao engenheiro de software como fazer uso de um processo definido de desenvolvimento de software, guiando-se pelos *scripts* que são providos pelo PSP. O processo definido de desenvolvimento, é muito importante para que seja possível a elaboração de produtos de software de qualidade. Também foram apresentadas ao engenheiro de software as vantagens de fazer planos e usar estes para o desenvolvimento de produtos de software. A utilização de planos apresenta várias melhorias ao processo de desenvolvimento, guiando o engenheiro de software para saber quais as tarefas a serem executadas, quem as fará e em que ordem elas serão executadas.

A necessidade de fazer estimativas está relacionada à qualidade dos planos de projeto a serem desenvolvidos. Com estimativas precisas de tempo e tamanho, é possível elaborar planos de projeto de acordo com a realidade do produto a ser desenvolvido.

Com o aprendizado do PSP1.1, o engenheiro de software aprende maneiras de elaborar estimativas de tempo de desenvolvimento e como organizar as tarefas do dia-a-dia. O PSP1.1 mostra como estimar o tempo gasto na realização de uma tarefa, com base em medições anteriores feitas em tarefas semelhantes. Neste nível, é possível analisar os compromissos e assumir somente aqueles que possam realmente ser cumpridos. Aqui o engenheiro de software também estuda a importância das estimativas para o planejamento e elaboração de cronogramas.

As revisões de projeto e código devem ser efetuadas com responsabilidade pelo engenheiro de software, buscando encontrar e corrigir todos os erros possíveis. A revisão pode ser um aliado a mais na busca constante de desenvolver produtos de software de alta qualidade.

Na execução da revisão de projeto e código o engenheiro de software deve usar uma lista de verificação contendo todos os itens que precisam ser observados neste

processo. Esta lista de verificação deve ser constantemente atualizada e modificada, de acordo com a necessidade. A inclusão ou modificação de itens desta lista deve ser sempre que for observada a necessidade de mudança para atingir os objetivos da revisão.

O PSP2.1 provê ao engenheiro de software um conjunto de modelos e formulários para auxílio no registro dos dados do projeto de software. Estes modelos são utilizados para registrar as especificações do projeto, de maneira a facilitar o desenvolvimento, acompanhamento e revisão do projeto.

O desenvolvimento de programas cada vez maiores e mais complexos faz surgir a necessidade de um método para auxílio no desenvolvimento destes. O PSP3 provê um método para desenvolvimento de grandes programas de forma cíclica. Dividindo grandes programas em partes menores e de fácil compreensão. Neste nível do PSP, o foco passa para o desenvolvimento de grandes projetos, embora, em nível pessoal. Este nível do PSP faz com que o engenheiro de software consiga aplicar os métodos de nível pessoal, a programas com mais de mil LOCs.

4 Uma Ferramenta de Suporte à Aplicação do PSP

4.1 INTRODUÇÃO

Neste capítulo iremos apresentar a proposta para ferramenta para automação do Personal Software Process - PSP. O objetivo deste sistema é auxiliar os engenheiros de software e gerentes de organizações de software na análise dos dados coletados no PSP. Através desta análise será possível descobrir onde o engenheiro precisa melhorar, quais os defeitos que o engenheiro injeta em um projeto e onde estes defeitos são removidos. Além disso a automação do PSP provê o armazenamento e a recuperação de dados de projetos prévios, que podem auxiliar no desenvolvimento de estimativas para novos projetos.

Conforme apresentado anteriormente, o elevado número de formulários a serem preenchidos e a grande quantidade de cálculos necessários para a transformação dos dados dos engenheiros em informações é uma das principais razões para o desenvolvimento deste. Além disso a coleta de dados de tempos e defeitos é simplificada com a automação, os cálculos necessários serão realizados automaticamente e de maneira mais fácil, não havendo necessidade de duplicação dos dados, tornando estes consistentes entre os formulários. Os dados pessoais estarão armazenados corretamente e sempre disponíveis para consultas e análises.

4.2 ANÁLISE DE REQUISITOS

4.2.1 Descrição do produto

O trabalho a ser desenvolvido é uma ferramenta para automação dos cálculos e análises de dados gerados pela utilização do PSP. Esta ferramenta irá auxiliar o engenheiro no registro e recuperação de seus dados pessoais providos pela utilização do PSP. Esta ferramenta possibilitará ao engenheiro a recuperação e análise segura dos seus dados pessoais, para que seja possível fazer a análise destes, possibilitando conhecer melhor a performance individual e constatando pontos do processo de desenvolvimento que precisam ser melhorados.

4.2.2 Objetivo do sistema

O sistema tem como objetivo implementar o suporte automatizado para o armazenamento e recuperação dos dados pessoais providos pela utilização do PSP, servindo como base de informações na busca da melhoria do processo pessoal. O sistema deverá ser capaz de prover ao engenheiro as informações necessárias para análise do processo pessoal, bem como, deverá facilitar o processo de transformação dos dados pessoais em informações úteis na avaliação do processo pessoal.

4.2.3 Objetivos Específicos do sistema

O sistema deverá prover os seguintes benefícios ao engenheiro de software:

- possibilidade de análise e verificação da consistência dos dados logo após a entrada dos dados no sistema;
- relatórios do processo de desenvolvimento, com dados do projeto e dados pessoais da performance do engenheiro de software;
- relatórios de defeitos, mostrando a fase onde foram injetados e removidos e tempo gasto na remoção;
- recuperação dos dados sobre propostas de melhorias do processo;
- visualização da fase do processo de desenvolvimento onde está sendo gasto a maior parte do tempo;
- visualização da fase onde está ocorrendo a injeção do maior número de defeitos;
- registro dos dados de projeto como estado dos objetos, cenários, lógica do programa;
- registro das revisões de projeto e código, para avaliação do desempenho destas;

4.2.4 Informações do sistema

O levantamento dos requisitos do sistema ocorre com a análise dos formulários, padrões, modelos e relatórios que são utilizados pelo PSP. Através deste levantamento foi possível definir alguns pontos que devem ser considerados na elaboração do sistema.

- O software deve manter o cadastro de sistemas que estão sendo desenvolvidos.
- Projeto PSP é o cadastro das informações (sistema, engenheiro, nível PSP etc) sobre o projeto que está sendo desenvolvido e analisado.
- O software ira manter o cadastro de engenheiro de software que estão trabalhando nos vários projetos existentes.
- O software deve ter o cadastro de níveis do PSP. Em todo o PSP são sete níveis e devem ser cadastrados à medida em que o engenheiro avança para os níveis superiores.
- Para cada sistema, engenheiro de software e nível do PSP, haverá um Projeto PSP. Cada Projeto PSP tem um Registro de Tempo, um Registro de Defeito e um Resumo de Plano de Projeto. Os demais itens de que compõe um Projeto PSP são disponibilizados de acordo com o nível do PSP.
- Os cadastros de Log de Tempo e Log de Defeito e o Resumo de Plano de Projeto são disponibilizados nos sete níveis do PSP.
- Os cadastros de Proposta de Melhoria do Processo e o Padrão de Código são disponibilizados a partir do segundo nível do PSP (PSP0.1).
- A partir do nível três do PSP (PSP1) são disponibilizados os cadastros de Instruções de Teste e Estimativa de Tamanho.
- Os cadastros de Modelo de Plano de Horário e Modelo de Plano de Tarefa ficam disponíveis ao usuário a partir do nível quatro do PSP (PSP1.1).
- Os cadastros do Checklist da Revisão de Projeto e CheckList da Revisão de Código são disponibilizados a partir do nível cinco do PSP (PSP2).
- A partir do nível seis do PSP (PSP2.1) são disponibilizados os cadastros do Modelo de Cenário Operacional, Modelo Especificação de Estado, Modelo Especificação Funcional e Modelo Especificação Lógica.

- E os cadastros de Resumo de Ciclo e do Log de Acompanhamento de Assunto são disponibilizados no último nível do PSP (PSP3).

4.3 REQUISITOS DO SISTEMA

Os requisitos identificados para este modelo, relacionam as atividades para execução de uma aplicação para utilização do PSP. Os seguintes itens foram identificados:

- Sistema
- Nível PSP
- Engenheiro de software
- Fase de desenvolvimento
- Projeto PSP
- Resumo de plano de projeto
- Padrões de defeitos
- Defeitos corrigidos
- Registro de Tempo
- Proposta de melhoria do processo
- Estimativa de tamanho
- Relatório de teste
- Lista de verificação de projeto
- Lista de verificação de código
- Modelo de especificação lógica
- Modelo de especificação de estado
- Modelo de especificação funcional
- Modelo de cenário operacional
- Log de acompanhamento de assunto
- Resumo de ciclo

4.4 DETALHAMENTO DO MODELO

O modelo será proposto utilizando Unified Modeling Language (Linguagem de Modelagem Unificada) – UML. Serão desenvolvidos os seguintes diagramas: Diagrama de Caso de Uso, Diagrama de Classe e Diagrama de Seqüência.

A modelagem de um diagrama de caso de uso é uma técnica usada para descrever os requisitos funcionais de um sistema. Os atores representam o papel de uma entidade externa ao sistema, por exemplo um usuário ou outro sistema que interage de alguma maneira com o sistema modelado.

Um **caso de uso** é um documento que apresenta a seqüência de passos ou cenários que um ator utiliza para completar um processo e atingir seu objetivo (JACOBSON et al., 1992). Um ator é conectado a um ou mais casos de uso através de associações, e tanto atores quanto casos de uso podem possuir relacionamentos de generalização que definem um comportamento comum de herança em superclasses especializadas em subclasses.

O **diagrama de classes** é muito importante nas metodologias orientadas a objetos. Um diagrama de classes descreve os tipos de objetos no sistema e os vários relacionamentos estáticos existentes entre eles. Os diagramas de classes também mostram atributos e operações de uma classe e as restrições de acordo com o modo com que os objetos são conectados (FOWLER & SCOTT, 2000).

O diagrama de classes apresenta a estrutura estática das classes de um sistema onde estas representam os objetos que são gerenciados pela aplicação modelada. Classes podem se relacionar com outras através de diversas maneiras: associação (conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe), ou em pacotes (classes agrupadas por características similares). O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. Um sistema normalmente possui mais de um diagrama de classes, já que não são todas as classes que estão inseridas em um único diagrama e uma classe pode participar de vários diagramas de classes.

O **diagrama de seqüência** apresenta a seqüência de interações de objetos organizados em uma seqüência de tempo, mostrando os objetos que participam na interação e a seqüência de mensagens trocadas (FURLAN, 1998). O diagrama de seqüência inclui a seqüência de tempo mas não inclui relacionamentos de objetos.

Um diagrama de seqüência apresenta a colaboração dinâmica entre os objetos de um sistema. O aspecto mais importante deste diagrama é que a partir dele percebe-se a seqüência de mensagens que são enviadas entre os objetos. Este diagrama mostra a interação entre os objetos, ou seja alguma coisa que acontecerá em um ponto específico da execução do sistema. O decorrer do tempo é visualizado observando-se o diagrama no sentido vertical de cima para baixo. As mensagens enviadas por cada objeto são simbolizadas por setas entre os objetos que se relacionam.

Os diagramas de seqüência podem mostrar objetos que são criados ou destruídos como parte do cenário documentado pelo diagrama. Um objeto pode criar outros objetos através de mensagens.

4.4.1 Especificação do Modelo

O projeto a ser desenvolvido faz uso da notação de Unified Modeling Language – UML, e será apresentada na seguinte ordem, iniciando com a apresentação do caso de uso, seguido do diagrama de classe e finalmente o diagrama de seqüência.

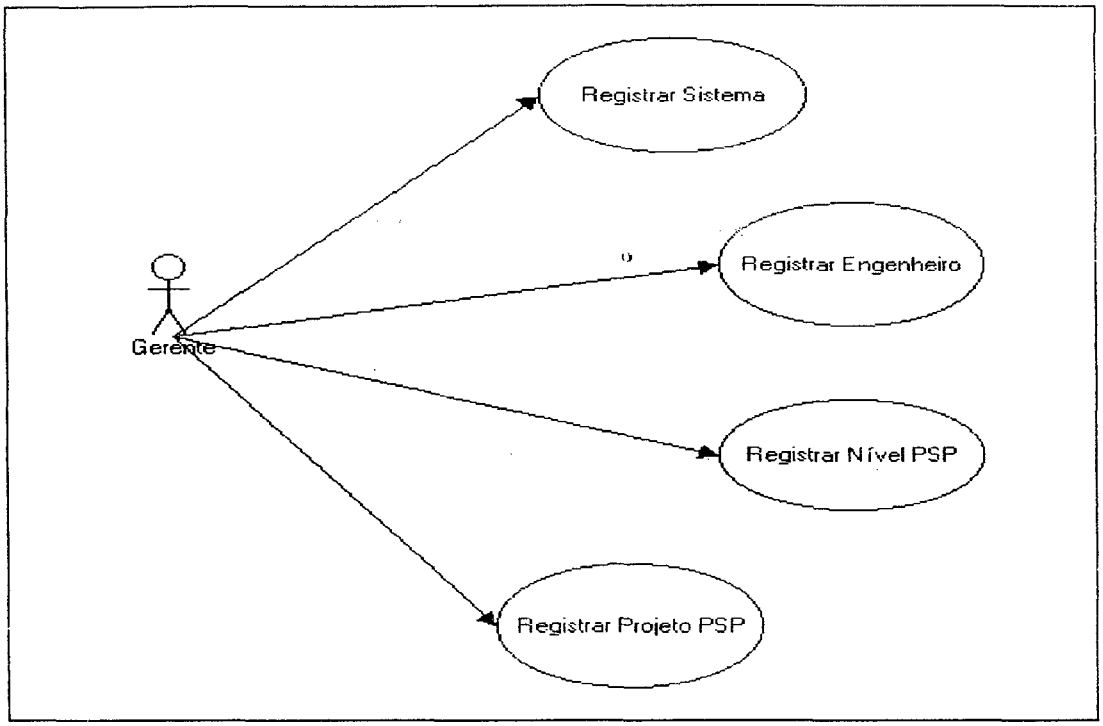
4.4.1.1 Registrar Projeto PSP

O registro de Projeto PSP, Figura 4.1, é de responsabilidade do ator Gerente. É ele que faz o cadastramento das tabelas necessárias e que são utilizadas pelos engenheiros de software na hora do registro dos dados pessoais. O primeiro cadastro é o Sistema, Quadro 4.1, ou seja, o nome do projeto no qual o engenheiro irá trabalhar. Outro cadastro mantido é o de Engenheiro, Quadro 4.3, é onde o gerente registra o nome e os dados dos engenheiros que estão trabalhando em algum projeto.

O cadastro de Nível do PSP, Quadro 4.5, também é muito importante, pois são cadastrados os níveis no quais os engenheiros podem trabalhar. O PSP tem sete níveis, à medida que é avançado para níveis maiores, aumenta o número de informações necessárias. E finalmente, o registro de Projeto PSP, Quadro 4.7. No Projeto PSP são armazenadas as informações do Sistema, Engenheiro, Nível do PSP, programa etc. Cada combinação entre Sistema, Engenheiro e Nível do PSP pode formar um Projeto PSP. Na

classe Projeto PSP, são armazenadas informações sobre o projeto, programa, data de início do projeto etc.

Figura 4.1 Diagrama de Caso de Uso "Registrar Projeto PSP"



Quadro 4.1 Caso de Uso Registrar Sistema

Caso de Uso	Registrar Sistema
Atores	Gerente
Finalidade	Inserir e manter as informações pertinentes ao sistema que está sendo desenvolvido.
Visão geral	Este caso de uso é iniciado pelo gerente, registrando os dados do sistema que está sendo desenvolvido.
Tipo	Secundário
Referências cruzadas	

Quadro 4.2 Seqüência Típica de Eventos para Registrar Sistema

Seqüência típica de eventos	
Ação do ator	Resposta do sistema

1. Gerente escolhe no menu a opção para fazer cadastro do sistema.	2. Abertura da janela para entrada dos dados pertinentes ao sistema.
3. Gerente informa os dados.	4. Confirmação de registro.
9. Gerente encerra sessão.	
Seqüências alternativas:	
- Linha 3: O Sistema já existe no banco de dados. O gerente pode alterar os dados existentes, informar outros dados ou encerrar a sessão.	

Quadro 4.3 Caso de Uso Registrar Engenheiro

Caso de Uso	Registrar Engenheiro
Atores	Gerente
Finalidade	Inserir e manter as informações pertinentes ao engenheiro que está trabalhando no sistema em desenvolvimento.
Visão geral	Este caso de uso é iniciado pelo gerente, registrando os dados do engenheiro que está desenvolvendo o sistema.
Tipo	Secundário
Referências cruzadas	

Quadro 4.4 Seqüência Típica de Eventos para Registrar Engenheiro

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Gerente escolhe no menu a opção para fazer cadastro do engenheiro.	2. Abertura da janela para entrada dos dados do engenheiro.
3. Gerente informa os dados.	4. Confirmação de registro.
9. Gerente encerra sessão.	
Seqüências alternativas:	
- Linha 3: O engenheiro já está cadastrado no banco de dados. O gerente pode alterar os dados existentes, informar outros dados ou encerrar a sessão.	

Quadro 4.5 Caso de Uso Registrar Nível do PSP

Caso de Uso	Registrar Nível do PSP

Atores	Gerente
Finalidade	Inserir e manter as informações sobre os níveis do PSP que são utilizados no processo de desenvolvimento.
Visão geral	Este caso de uso é iniciado pelo gerente, registrando os dados do Nível do PSP que será usado no desenvolvimento.
Tipo	Secundário
Referências cruzadas	

Quadro 4.6 Seqüência Típica de Eventos para Registrar Nível PSP

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Gerente escolhe no menu a opção para fazer cadastro do Nível do PSP	2. Abertura da janela para entrada dos dados do Nível do PSP.
3. Gerente informa os dados.	4. Confirmação de registro.
9. Gerente encerra sessão.	
Seqüências alternativas:	
- Linha 3: O Nível do PSP já existe no banco de dados. O gerente pode alterar os dados existentes, informar outros dados ou encerrar a sessão.	

Quadro 4.7 Descrição Caso de Uso Registrar Projeto PSP

Caso de Uso	Registrar Projeto PSP
Atores	Gerente
Finalidade	Inserir e manter as informações pertinentes a novos projetos. Para cada Sistema, Engenheiro e Nível PSP, pode ser criado um Projeto PSP.
Visão geral	Este caso de uso é iniciado pelo gerente, escolhendo o sistema a ser desenvolvido, qual o engenheiro que trabalhará no sistema e qual o nível do PSP que será utilizado.
Tipo	Primário
Referências cruzadas	Registrar Sistema, Registrar Engenheiro e Registrar Nível PSP.

Quadro 4.8 Seqüência Típica de Eventos para Registrar Projeto PSP

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Gerente escolhe opção menu para fazer o cadastro do Projeto PSP.	2. Abertura da janela para entrada dos dados do Nível do PSP.
3. Gerente seleciona Sistema.	4. Sistema selecionado.
5. Gerente seleciona Engenheiro.	6. Engenheiro selecionado.
7. Gerente seleciona Nível PSP.	8. Nível do PSP selecionado.
9. Gerente informa outros dados do projeto e confirma o registro do Projeto PSP.	10. Confirmação de registro.
11. Gerente encerra sessão.	
Seqüências alternativas:	
- Linha 1: O sistema informado não existe no banco de dados. O gerente informa outros dados ou encerra a sessão. - Linha 3: O engenheiro informado não existe no banco de dados. O gerente informa outros dados ou encerra a sessão. - Linha 5: O nível do PSP informado não existe no banco de dados. O gerente informa outros dados ou encerra a sessão. - Linha 7: O Projeto PSP (Sistema, Engenheiro, Nível PSP) já existe no sistema. O gerente pode informar outros dados ou encerrar a sessão.	

O diagrama de classe, Figura 4.2, mostra um relacionamento ternário, que é a associação entre três ou mais classes. Cada Projeto PSP tem um Sistema, um Engenheiro e um Nível PSP.

O sistema foi projetado desta maneira para permitir que um engenheiro trabalhe em mais de um sistema paralelamente. Neste modelo, o engenheiro pode trabalhar em vários Níveis do PSP, no mesmo sistema ou até mesmo em sistema diferentes.

Figura 4.2 Diagrama de Classe Registrar Projeto PSP

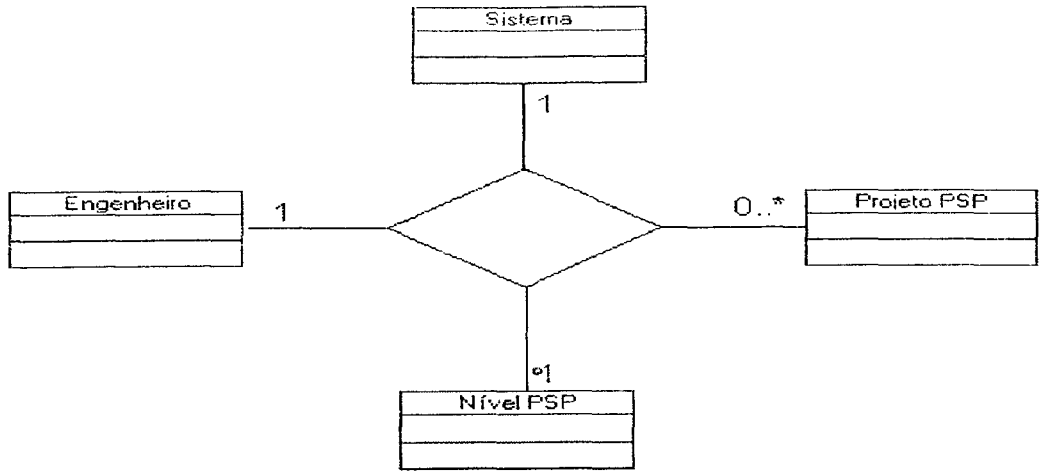
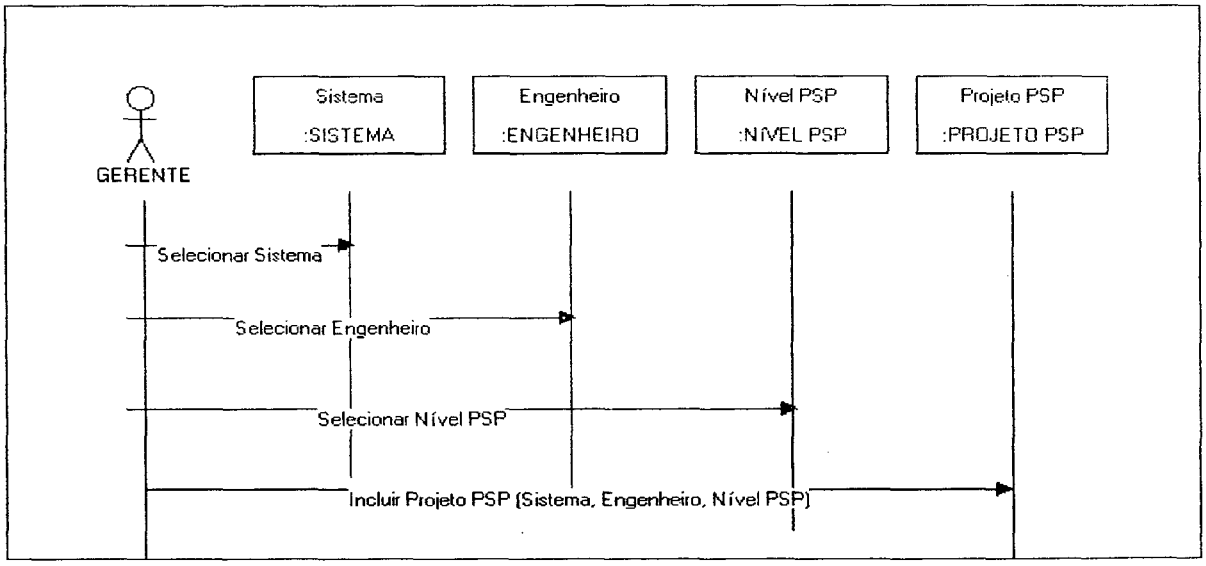


Figura 4.3 Diagrama de Seqüência do caso de uso Registrar Projeto PSP



O diagrama de seqüência, Figura 4.3, apresenta a seqüência de ações a serem seguidas para o registro de Projeto PSP, sendo necessário existir um Sistema, um Engenheiro e um Nível PSP.

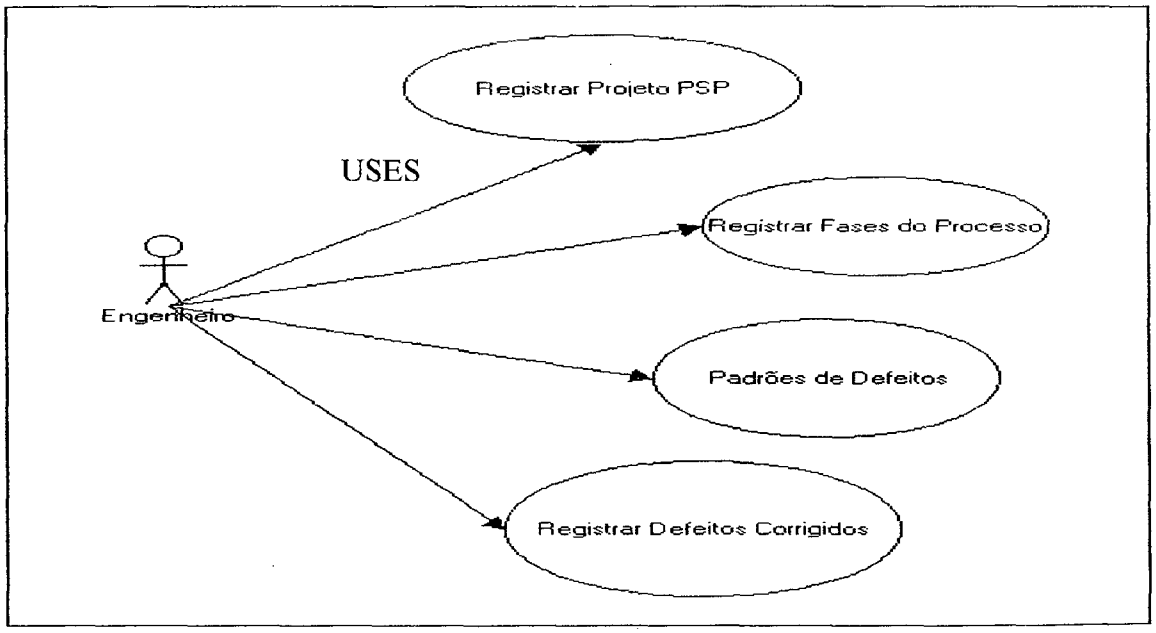
4.4.1.2 Registrar Defeitos Corrigidos

Este é um caso de uso muito importante durante o processo de utilização do PSP. Aqui o engenheiro registra todos os defeitos encontrados e corrigidos durante o processo de desenvolvimento do software. O processo de software é dividido em fases de desenvolvimento, então há necessidade de registrar e manter todas as fases do processo, Quadro 4.9. O PSP define alguns tipos de defeito padrão, porém pode haver necessidade do registro de outros tipos. O cadastro de tipos de defeito padrão é apresentado no Quadro 4.11. Os defeitos corrigidos, Quadro 4.13, devem ser registrados para que seja possível identificar quais os erros que são injetados, onde estes acontecem e onde são removidos.

No registro dos erros corrigidos é muito importante fazer o registro da fase onde o erro foi injetado e da fase onde o erro foi corrigido. Para que seja possível atingir melhores resultados na avaliação do processo, é necessário que todos os erros sejam registrados.

O caso de uso Registrar Projeto PSP, é descrito no item anterior.

Figura 4.1 Diagrama de Caso de Uso "Registrar Defeitos Corrigidos"



Quadro 4.1 Caso de Uso Registrar Fases do Processo

Caso de Uso	Registrar Fase do Processo
Atores	Engenheiro
Finalidade	Inserir e manter as informações das fases do processo de

	desenvolvimento.
Visão geral	Este caso de uso é iniciado pelo engenheiro, registrando os dados das fases do processo.
Tipo	Secundário
Referências cruzadas	

Quadro 4.2 Seqüência Típica de Eventos para Registrar Fases do Processo

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Engenheiro escolhe no menu a opção para fazer cadastro das fases do processo	2. Abertura da janela para entrada dos dados pertinentes a fase do processo.
3. Engenheiro informa os dados.	4. Confirmação de registro.
9. Engenheiro encerra sessão.	
Seqüências alternativas:	
- Linha 3: A fase já está cadastrada no banco de dados. O engenheiro pode alterar os dados existentes, informar outros dados ou encerrar a sessão.	

Quadro 4.3 Caso de Uso Registrar Tipos de Defeitos

Caso de Uso	Registrar Tipos de Defeitos
Atores	Engenheiro
Finalidade	Inserir e manter as informações referentes aos tipos de defeitos.
Visão geral	Este caso de uso é iniciado pelo engenheiro, registrando os dados dos tipos de defeitos que podem ser cometidos pelo engenheiro.
Tipo	Secundário
Referências cruzadas	

Quadro 4.4 Seqüência Típica de Eventos para Registrar Tipos de Defeitos

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Engenheiro escolhe no menu a opção para fazer cadastro de tipos de defeitos.	2. Abertura da janela para entrada dos dados pertinentes ao tipo de defeitos.

3. Engenheiro informa os dados.	4. Confirmação de registro.
9. Engenheiro encerra sessão.	
Seqüências alternativas:	
- Linha 3: O tipo de defeito já está cadastrado no banco de dados. O engenheiro pode alterar os dados existentes, informar outros dados ou encerrar a sessão.	

Quadro 4.5 Descrição Caso de Uso Registrar Defeitos Corrigidos

Caso de Uso	Registrar defeitos corrigidos
Atores	Engenheiro
Finalidade	Inserir e manter as informações pertinentes ao registro dos defeitos encontrados e corrigidos durante o desenvolvimento do sistema. Cada Projeto PSP pode ter vários registros de defeitos corrigidos.
Visão geral	Este caso de uso é iniciado pelo engenheiro de software, quando este deseja fazer o registro de todos defeitos encontrados e corrigidos.
Tipo	Primário
Referências cruzadas	Registrar Projeto PSP, Registrar Fase do Processo, Registrar Padrão de Defeitos.

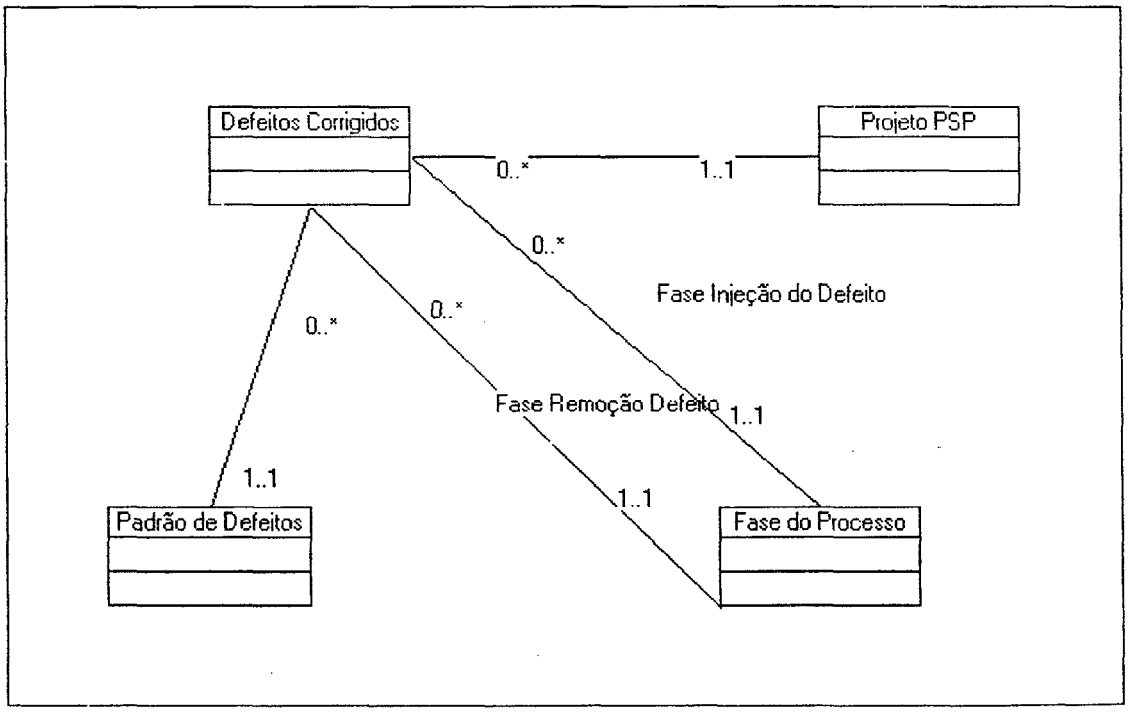
Quadro 4.6 Seqüência Típica de Eventos para Registrar Defeitos Corrigidos

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Engenheiro seleciona Projeto PSP	2. Projeto PSP selecionado.
3. Engenheiro seleciona Fase do processo.	4. Fase do processo selecionada.
5. Engenheiro seleciona Padrão de Defeitos.	6. Tipo de defeito selecionado.
7. Engenheiro informa dados sobre os defeitos e registra os defeitos.	8. Confirmação de registro.
9. Engenheiro encerra sessão.	

Seqüências alternativas:
- Linha 1: O engenheiro informa um Projeto PSP inexistente. O engenheiro pode informar outros dados ou encerrar a sessão. - Linha 3: O engenheiro informa uma Fase do Processo inexistente. O engenheiro pode informar outros dados ou encerrar a sessão. - Linha 5: O engenheiro informa um Padrão de Defeito inexistente. O engenheiro pode informar outros dados ou encerrar a sessão. - Linha 7: O engenheiro informa um defeito que já cadastrado. O engenheiro pode informar outros dados ou encerrar a sessão.

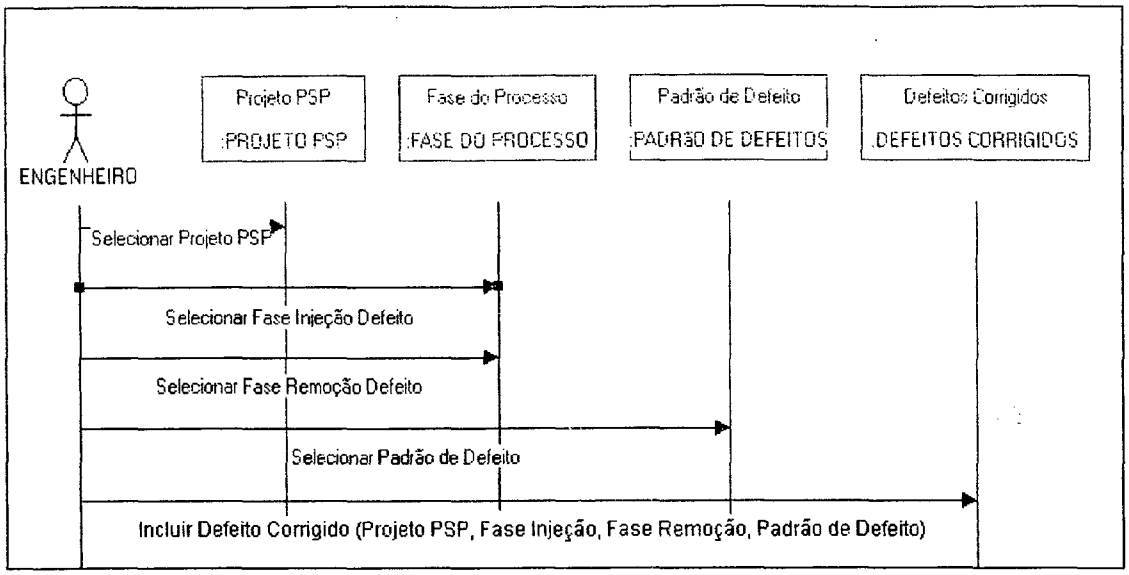
A Figura 4.5, apresenta o Diagrama de Classe Defeitos Corrigidos, mostrando as classes que são utilizadas no modelo para registro dos defeitos corrigidos.

Figura 4.2 Diagrama de Classe Defeitos Corrigidos



A Figura 4.6, apresenta a seqüência típica de eventos a serem efetuados para o registro dos defeitos corrigidos.

Figura 4.3 Diagrama de Seqüência Defeitos Corrigidos



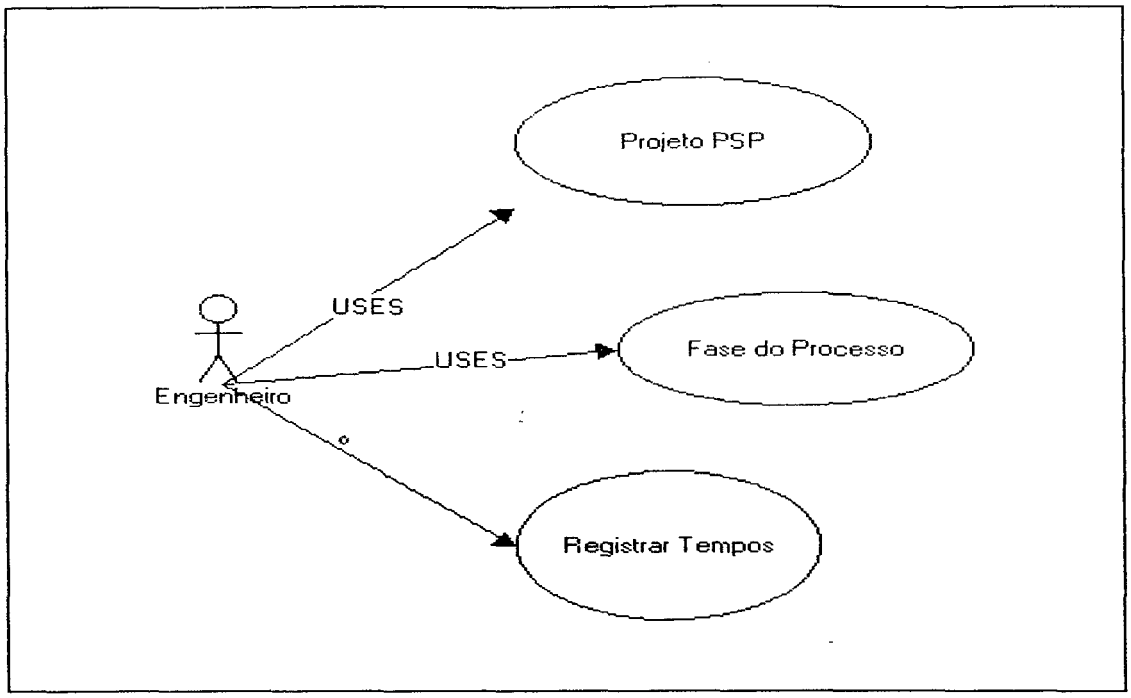
4.4.1.3 Registrar Tempos

O registro dos tempos do engenheiro é um caso de uso muito importante para o processo de utilização do PSP. Aqui o engenheiro registra o tempo de duração de cada tarefa executada, inclusive as interrupções que ocorrem. Para que o engenheiro consiga fazer planos de desenvolvimento há a necessidade de identificar como é gasto o seu tempo de trabalho. O tempo é registrado em minutos para que seja possível registrar todos os tempos de execuções, incluindo aquelas de menor tempo de duração.

Cada tarefa realizada pertence a uma fase do processo de desenvolvimento, então há necessidade de registrar e manter todas as fases do processo. No registro dos tempos é muito importante fazer o registro de cada tarefa executada, por menor que seja o tempo gasto na execução desta.

O caso de uso Registrar Projeto PSP e Fases do Processo são descritos anteriormente.

Figura 4.1 Diagrama do caso de uso Registrar Tempos



Quadro 4.1 Descrição Caso de Uso Registrar Tempos

Caso de Uso	Registrar tempos
Atores	Engenheiro
Finalidade	Inserir e manter as informações pertinentes ao registro dos tempos gastos nas diversas atividades desempenhadas pelo engenheiro de software. Cada Projeto PSP pode ter vários registros de tempos.
Visão geral	Este caso de uso é iniciado pelo engenheiro de software, quando este deseja fazer o registro do tempo gasto nas atividades.
Tipo	Primário
Referências cruzadas	Registrar Projeto PSP, Registrar Fase do Processo.

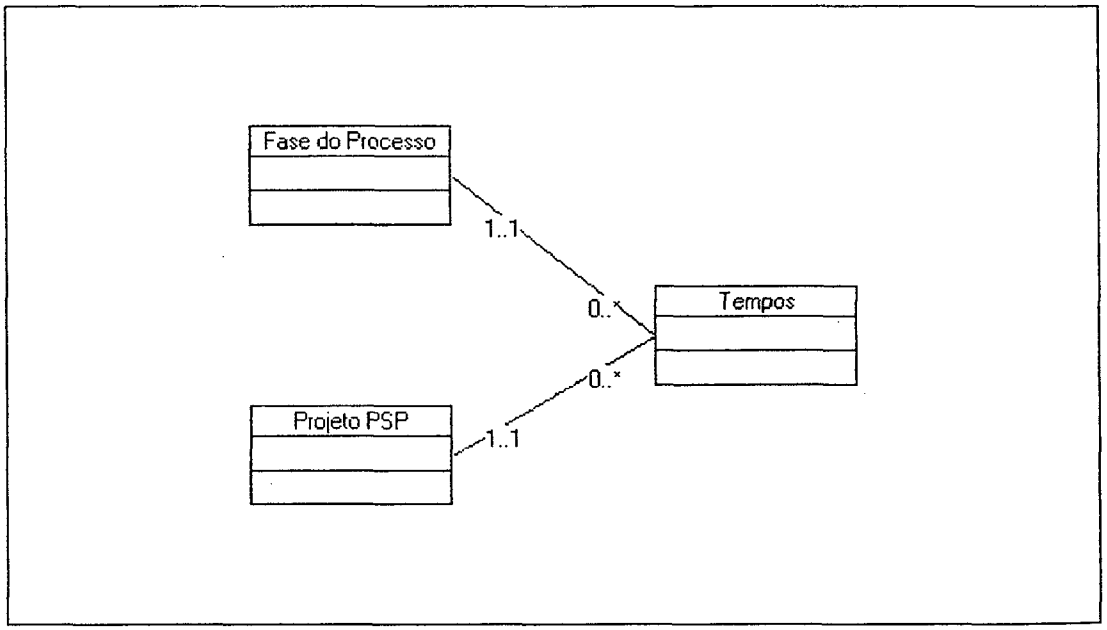
Quadro 4.2 Seqüência Típica de Eventos para Registrar Tempos

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Selecionar Projeto PSP	2. Projeto PSP selecionado.

3. Engenheiro registra Fase do processo.	4. Confirmação de registro.
5. Engenheiro informa dados sobre os tempos das tarefas.	6. Confirmação de registro.
7. Engenheiro encerra sessão.	
Seqüências alternativas:	
- Linha 1: O engenheiro informa um Projeto PSP inexistente. O engenheiro pode informar outros dados ou encerrar a sessão. - Linha 3: O engenheiro informa uma Fase do Processo inexistente. O engenheiro pode informar outros dados ou encerrar a sessão. - Linha 7: O engenheiro informa um Tempo já existente. O engenheiro pode informar outros dados ou encerrar a sessão.	

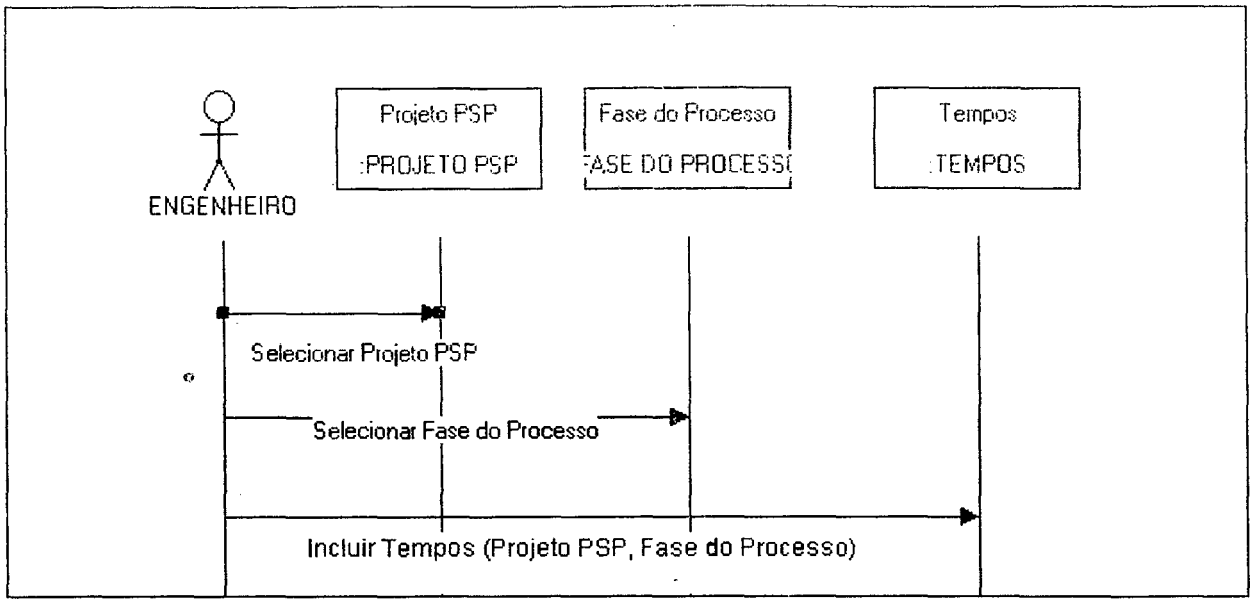
A Figura 4.8, apresenta o Diagrama Registro de Tempos, mostrando as classes que são utilizadas no modelo para registro de tempos gastos no desenvolvimento do projeto.

Figura 4.2 Diagrama de Classes Registro de Tempos



A Figura 4.9, apresenta a seqüência típica de eventos que devem ser seguidos para o registro dos tempos gastos durante o desenvolvimento do projeto.

Figura 4.3 Diagrama de Seqüência Registro de Tempos

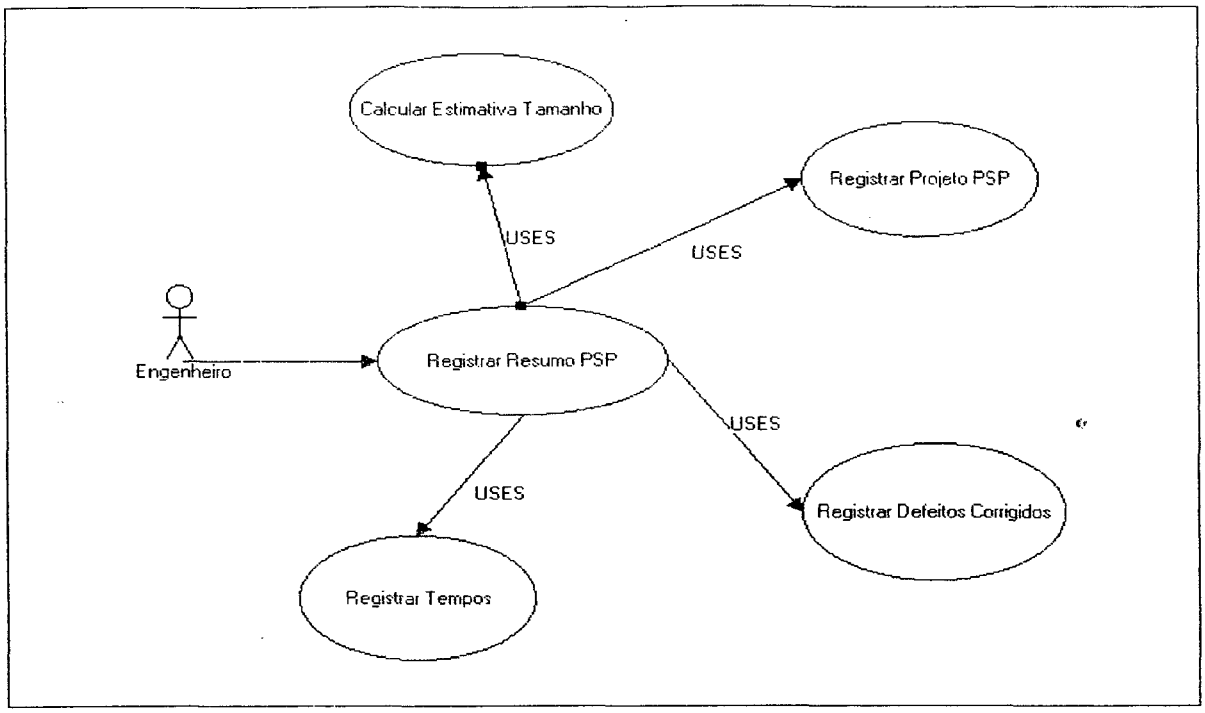


4.4.1.4 Registrar Resumo PSP

Para que seja possível analisar os dados de maneira produtiva, há a necessidade de ter os dados apresentados de maneira que seja fácil de entendê-los. O Resumo PSP provê esta maneira, fornecendo a recuperação de maneira fácil dos dados necessários para a análise de performance do engenheiro. Ao final de cada tarefa, o engenheiro deve completar o Resumo PSP com os dados pessoais para que possam ser utilizados em análises de desempenho e para auxilio no planejamento de projetos futuros.

O formulário de resumo de plano de projeto contém dados do engenheiro de software, o tempo total planejado, os tempos atuais do processo e o número de defeitos injetados e removidos em cada uma das fases do processo. Este formulário é preenchido na fase de *postmortem*, com os dados oriundos do log de registro de tempo e do log de registro de defeitos e alguns dados que são informados pelo próprio engenheiro. Os dados do tempo gasto na execução das tarefas, bem como os defeitos corrigidos, serão recuperados automaticamente do registro de defeitos corrigidos e dos tempos digitados.

Figura 4.1 Diagrama de caso de Registrar Resumo Projeto



Quadro 4.1 Descrição Caso de Uso Registrar Resumo Projeto

Caso de Uso	Registrar Resumo Projeto
Atores	Engenheiro
Finalidade	Inserir e manter as informações pertinentes ao resumo do projeto como por exemplo (registro do tempo gasto em cada fase, fase na qual os defeitos são injetados e corrigidos, tamanho do programa, eficiência do engenheiro, obtida através do número de defeitos do sistema, eficiência na remoção de defeitos). As informações variam de acordo com o nível do PSP. À medida em que se está avançando nos níveis, aumenta o número de informações.
Visão geral	Este caso de uso é iniciado pelo engenheiro de software, quando este deseja fazer o resumo dos dados de um programa em um determinado nível do PSP.
Tipo	Primário
Referências cruzadas	Registrar Projeto PSP, Registrar Tempos, Registrar Defeitos corrigidos.

Quadro 4.2 Seqüência Típica de Eventos para Registrar Resumo Projeto

Seqüência típica de eventos	
Ação do ator	Resposta do sistema
1. Selecionar Projeto PSP	2. Projeto PSP selecionado.
3. Para o projeto PSP selecionado, seleciona-se os defeitos corrigidos.	4. Defeitos corrigidos selecionados.
5. Para o projeto PSP selecionado, seleciona-se os tempos.	6. Tempos selecionados.
7. Engenheiro registra resumo do projeto.	8. Confirmação de registro.
7. Engenheiro encerra sessão.	
Seqüências alternativas:	
- Linha 1: O engenheiro informa um Projeto PSP inexistente. O engenheiro pode informar outros dados ou encerrar a sessão.	

Figura 4.2 Diagrama de Classe Resumo Projeto

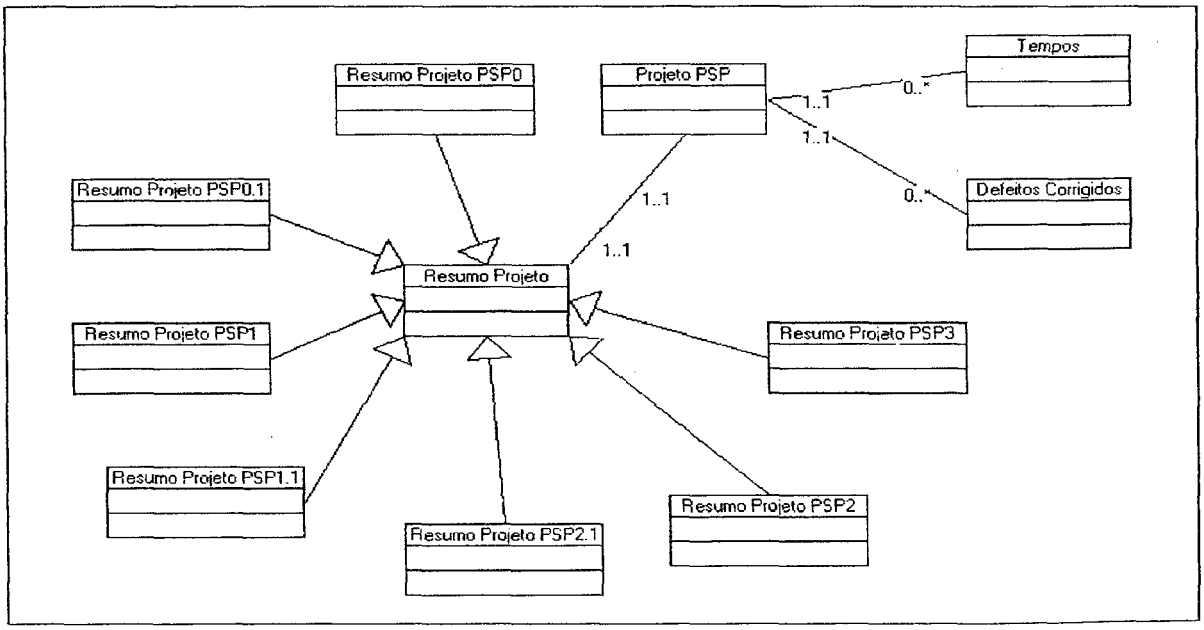
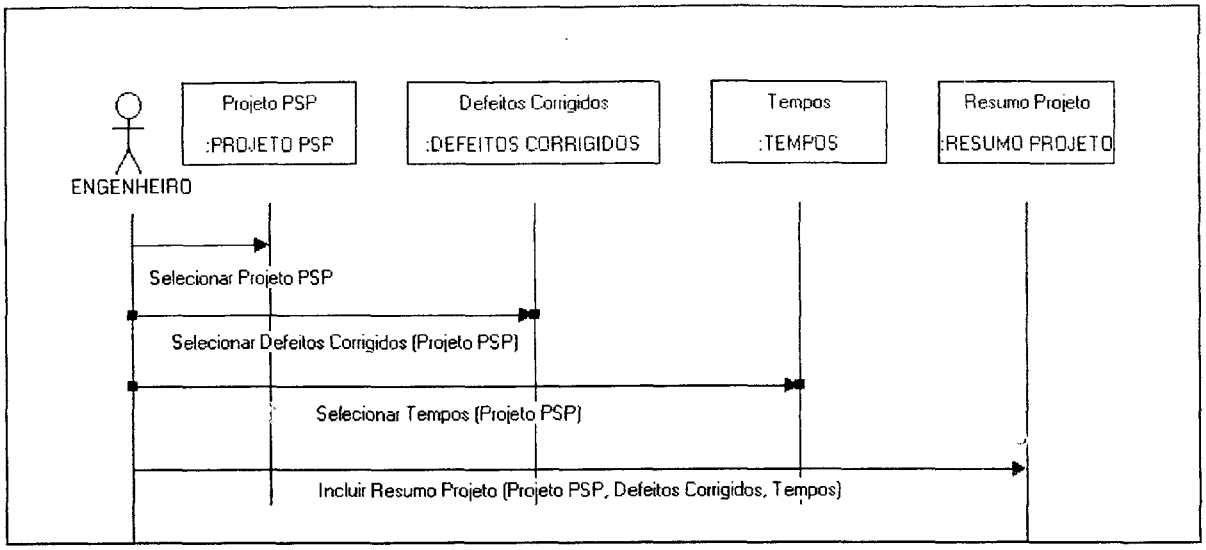


Figura 4.3 Diagrama de seqüência Resumo Projeto

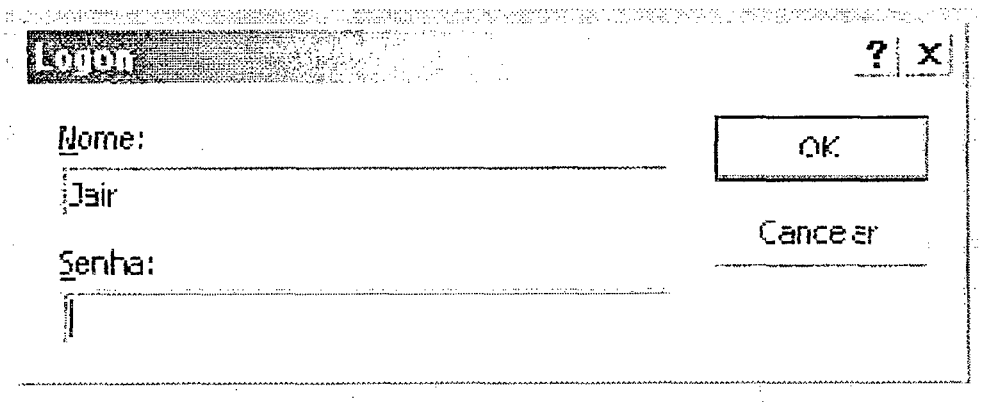


4.4.2 Implantação do sistema

As informações geradas através do levantamento dos requisitos, possibilitaram o desenvolvimento do sistema. Para tanto, foi utilizado o software Microsoft Access, como banco de dados e gerador de interface do cliente. Foram criadas as principais janelas do sistema, contendo os dados iniciais de cadastro, registro de tempo, registro de defeitos corrigidos e o registro do PSP nível 0. Estas janelas são apresentadas a seguir.

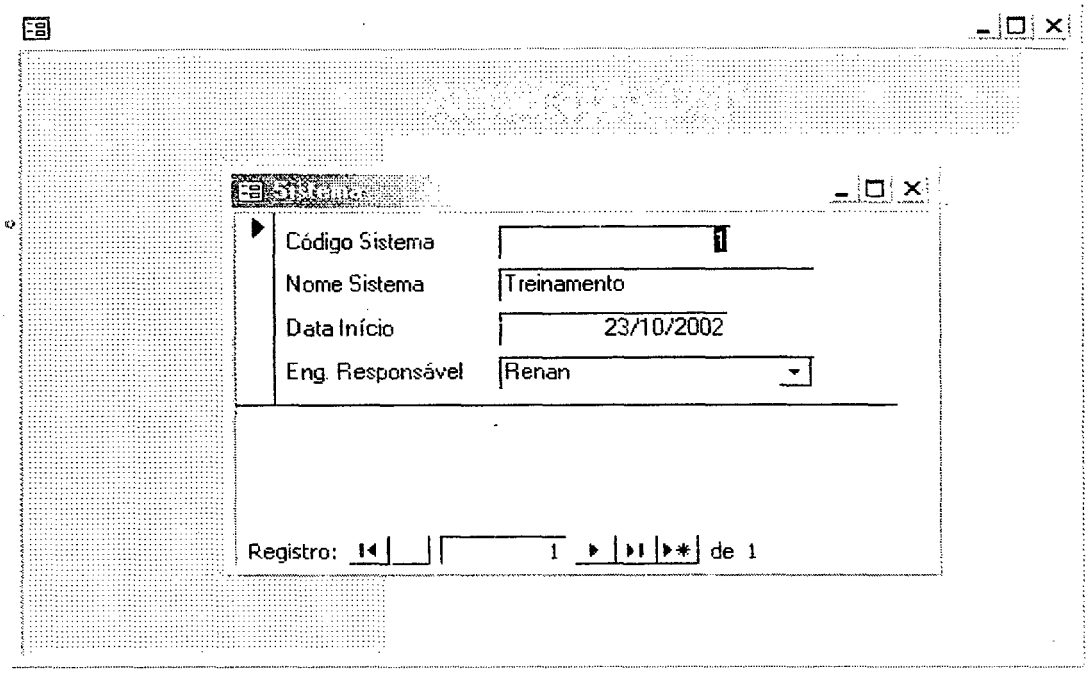
A Figura 4.13 apresenta a tela de Login no Sistema, onde o usuário informa seu nome e a sua senha de acesso ao sistema.

Figura 4.1 Login no Sistema



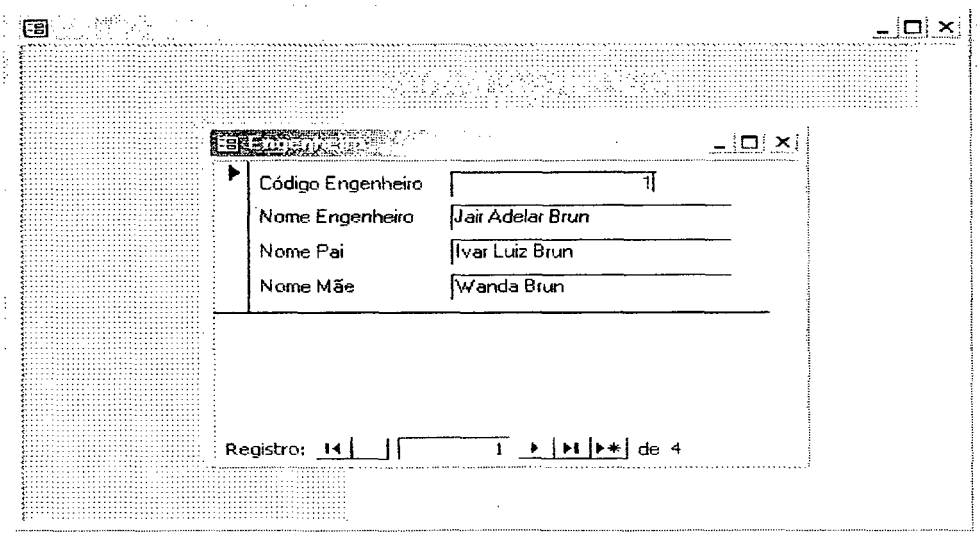
A Figura 4.14 apresenta a tela de registro do sistema que está sendo desenvolvido. Deve ser informados o nome do sistema, a data de início e o engenheiro responsável pela execução do sistema.

Figura 4.2 Registrar Sistema



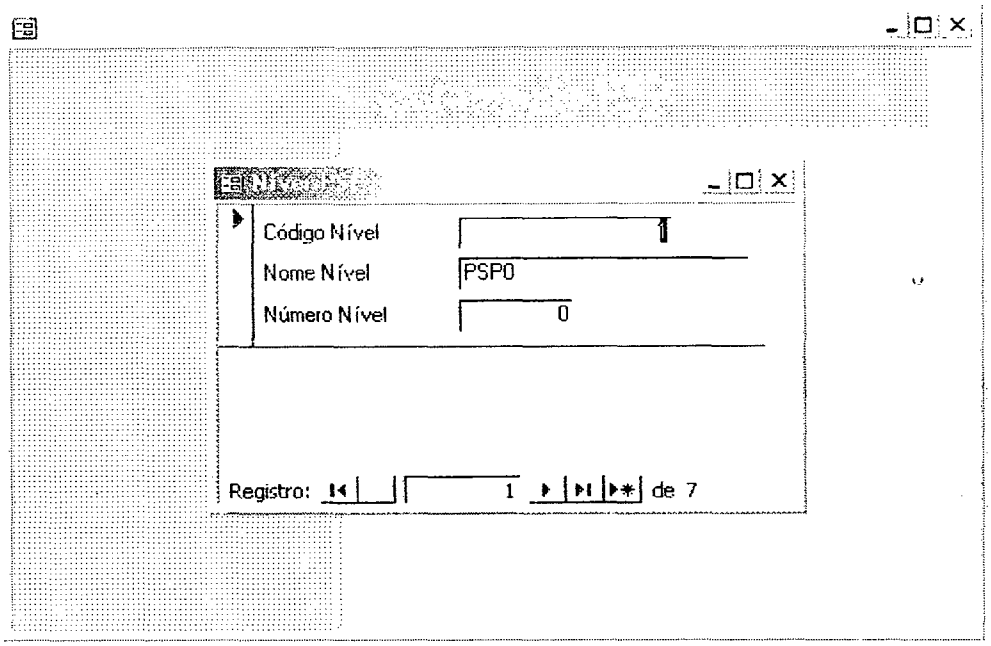
A Figura 4.15 apresenta a tela de registro do engenheiro. Sendo necessário informar o nome do engenheiro.

Figura 4.3 Registrar Engenheiro



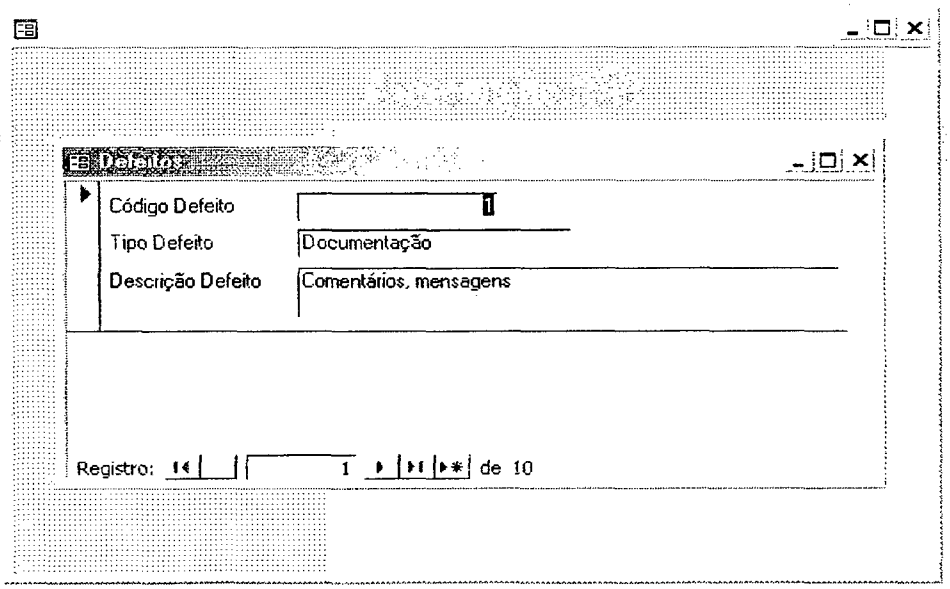
A Figura 4.16 apresenta a tela de registro do nível do PSP. Deve ser informados o nome do nível e o número deste (de 0 até 6).

Figura 4.4 Registrar Nível PSP



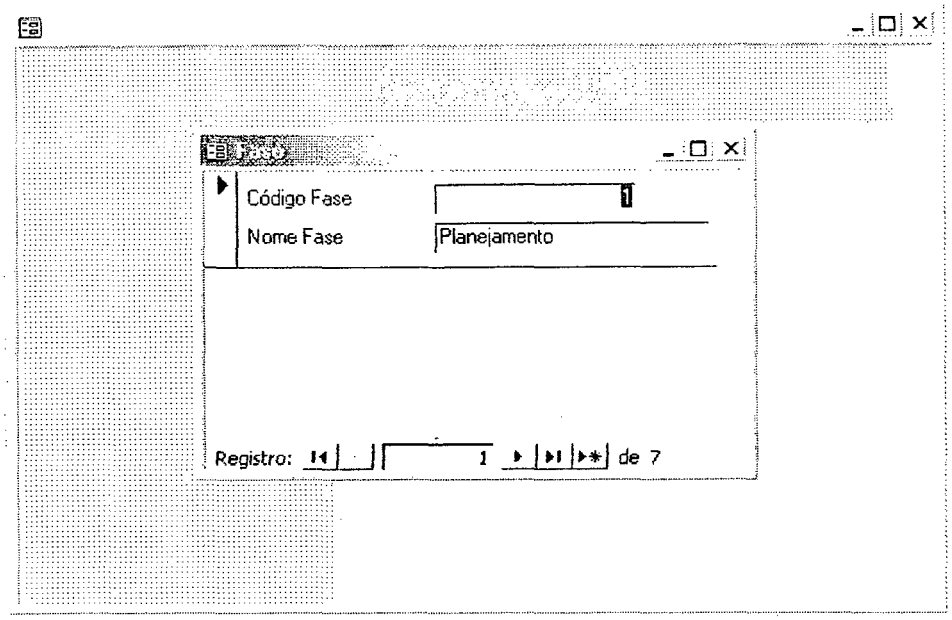
A Figura 4.17 apresenta a tela de registro dos possíveis defeitos injetados e removidos. Deve ser informados o tipo do defeito e a descrição deste.

Figura 4.5 Registrar Tipos de Defeitos



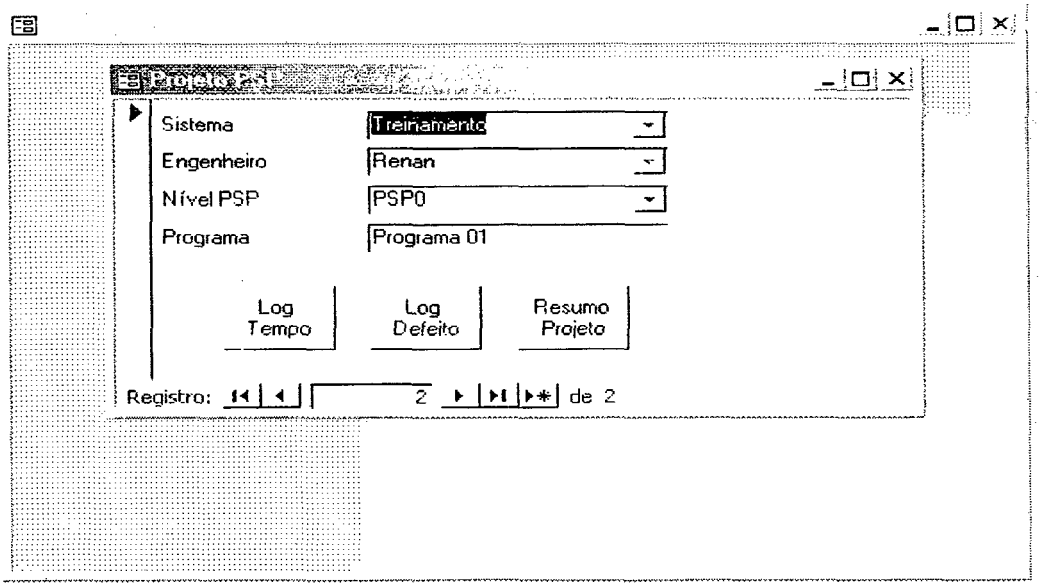
A Figura 4.18 apresenta a tela de registro das fases de desenvolvimento. Deve ser informado o nome da fase de desenvolvimento.

Figura 4.6 Registrar Fases do Processo



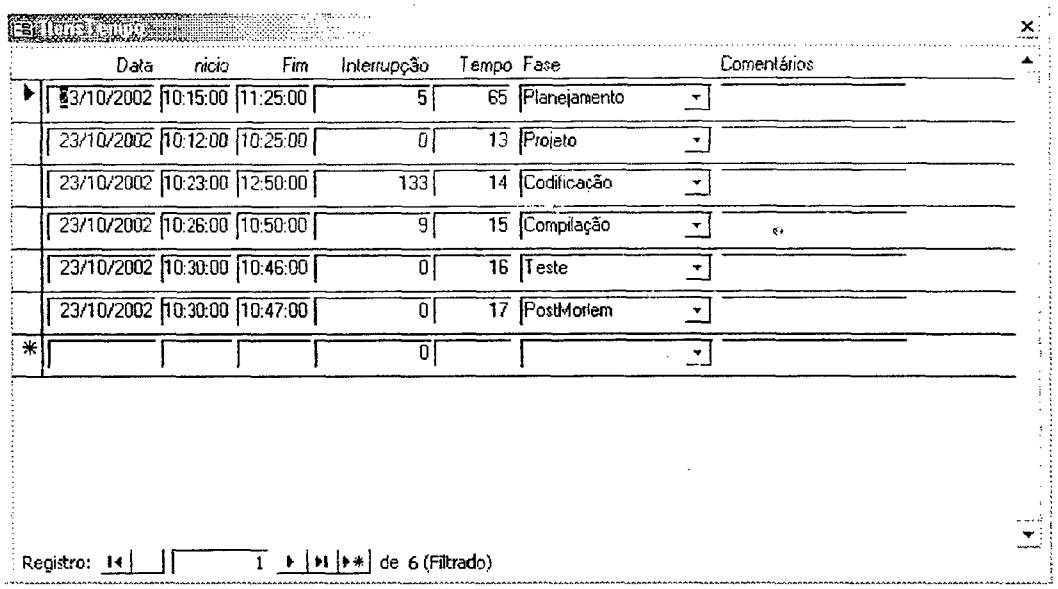
A Figura 4.19 apresenta a tela de registro do projeto desenvolvido. Deve ser informado o sistema, o engenheiro o nível do PSP e nome do programa.

Figura 4.7 Registrar Projeto PSP



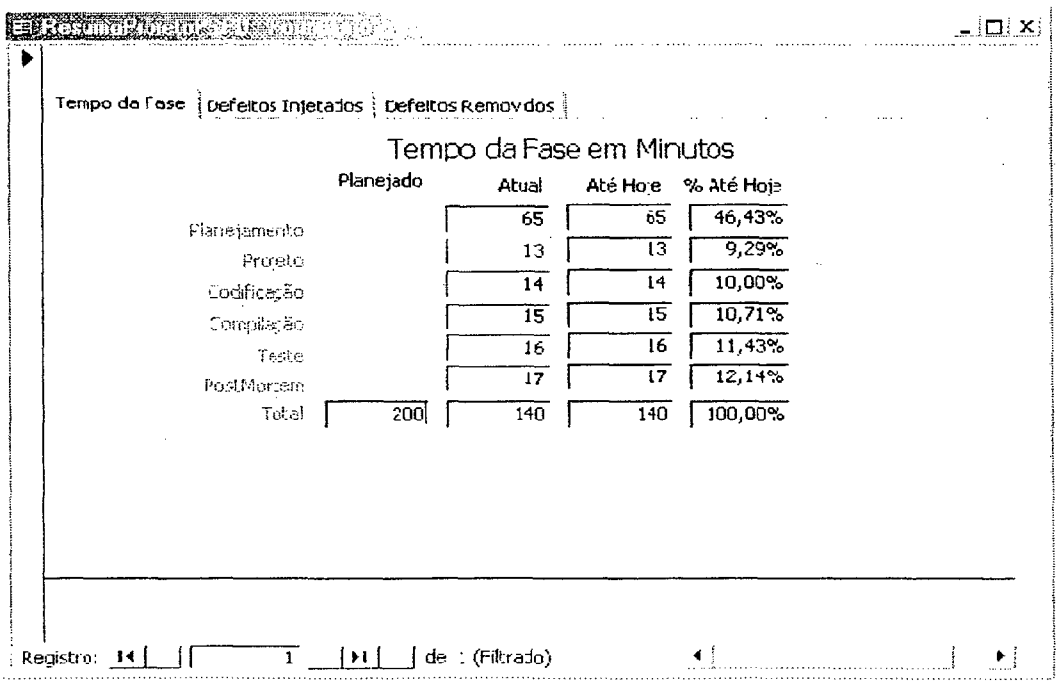
A Figura 4.20 apresenta a tela de registro dos Tempos. Deve ser informada a data, hora de início, hora final, tempo de interrupção, tempo gasto (hora final – hora de início – tempo de interrupção, a fase na qual se está trabalhando e comentários úteis.

Figura 4.8 Registrar Tempos



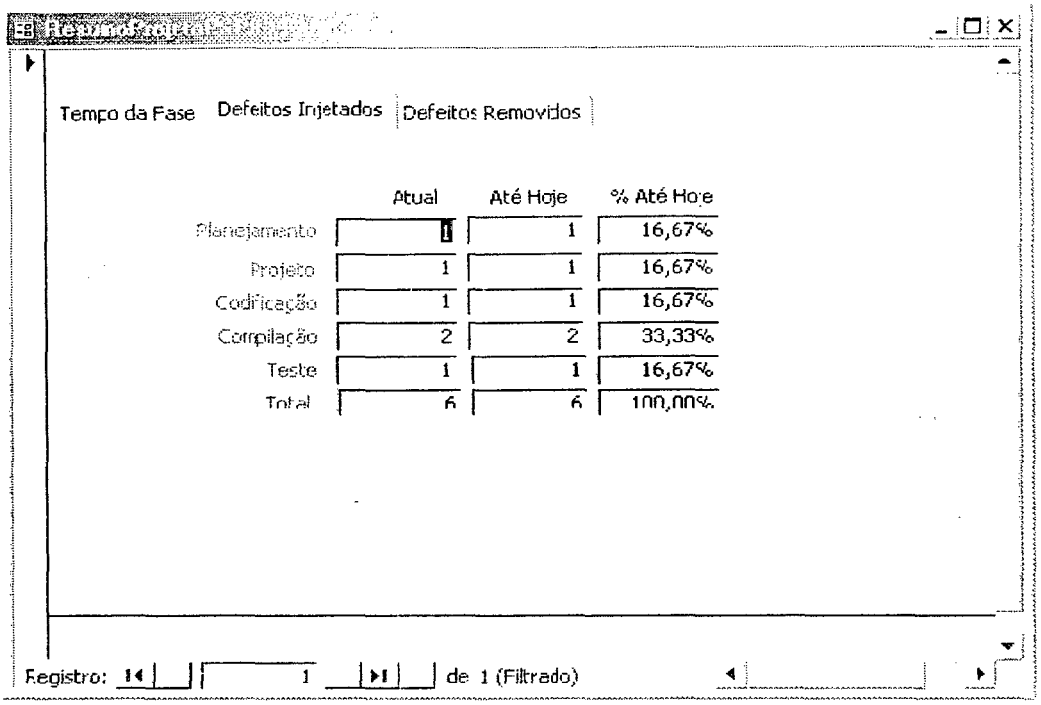
A Figura 4.21 apresenta a tela Tempo nas Fases. Aqui é apresentado um resumo do tempo gasto em cada uma das fases do processo de desenvolvimento.

Figura 4.9 Resumo PSP0 - Tempo nas Fases



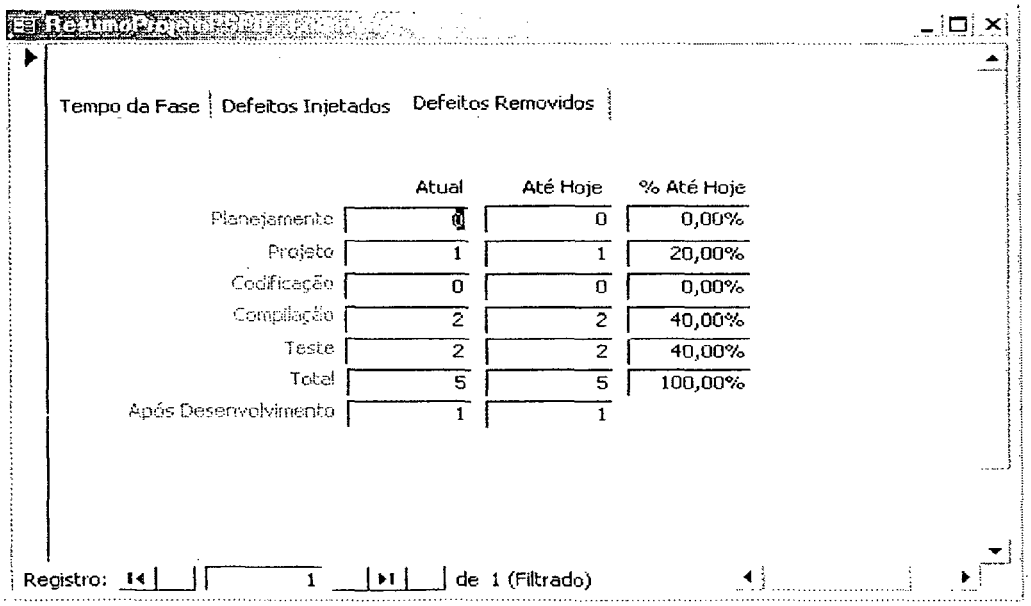
A Figura 4.22 apresenta a tela de Defeitos Injetados Aqui é apresentado um resumo dos defeitos injetados em cada uma das fases do processo de desenvolvimento.

Figura 4.10 Resumo PSP0 - Defeitos Injetados



A Figura 4.23 apresenta a tela de Defeitos Removidos Aqui é apresentado um resumo dos defeitos removidos em cada uma das fases do processo de desenvolvimento.

Figura 4.11 Resumo PSP0 - Defeitos Removidos



4.4.3 Descrição das Classes e atributos

A seguir é apresentada uma descrição das classes e atributos que compõe o sistema.

Quadro 4.1 Classe Padrão de Defeitos

CdPadrãoDefeito	Código do Padrão de Defeito
TipoDefeito	Tipo do Defeito
Descrição	Descrição para identificação do defeito

Quadro 4.2 Classe Engenheiro

cdEngenheiro	Código do Engenheiro
NomeEngenheiro	Nome do Engenheiro
NomePai	Nome do Pai
NomeMae	Nome da Mãe

Quadro 4.3 Classe Fase

cdFase Do Processo	Código da Fase do Processo
Nome da Fase	Nome da Fase do Processo

Quadro 4.4 Classe Defeitos Corrigidos

CdDefeitosCorrigidos	Código do Defeito Corrigido
Seqüência	Número seqüencial do defeito
DataCorreção	Data de Correção do Defeito
CdPadrãoDefeito	Defeito corrigido
CdFase do Processo Injetado	Fase do Processo em que foi injetado o defeito
CdFase do Processo Removido	Fase do Processo em que foi removido o defeito
TempoCorreção	Tempo Total gasto na correção do defeito
SeqüênciaInjetado	Seqüência onde o defeito foi injetado
Descrição	Descrição da correção do defeito

Quadro 4.5 Classe Tempo

CdTempo	Código dos Tempos
DataDaTarefa	Data de execução da tarefa
HoraInicio	Hora de inicio da tarefa
HoraFim	Hora de final da tarefa
TempoDeInterrupção	Tempo total de interrupções durante a execução
TempoDelta	(HoraFim – HoraInicio) – TempoDeInterrupção
Fase do Processo	Fase na qual foi gasto o tempo
Comentários	Comentários diversos ou sobre as interrupções

Quadro 4.6 Classe Níveis PSP

CdNível	Código do Nível
Descrição	Descrição do Nível
Seqüência	Número Seqüencial

Quadro 4.7 Classe Sistema

CdSistema	Código do Sistema
NomeSistema	Nome do sistema
DataInicio	Data de Inicio do desenvolvimento
Responsável	Nome do Engenheiro responsável

5 CONCLUSÕES E PERSPECTIVAS PARA TRABALHOS FUTUROS

5.1 CONCLUSÃO

A utilização de processos definidos para o desenvolvimento de software proporciona benefícios em relação à qualidade do produto elaborado. O *Personal Software* Process - PSP é considerado por algumas pessoas como o estado da arte na engenharia de software. Provavelmente é o processo de desenvolvimento mais estruturado e medido, desenvolvido pelo SEI (RICO, 2000).

A adoção do PSP pode proporcionar benefícios para o engenheiro de software, pois provê dados para que seja feita a avaliação de suas forças e fraquezas. Pontos fortes e pontos fracos que o engenheiro possui no processo de desenvolvimento de software. Estes dados, usados adequadamente, podem guiar o engenheiro de software na melhoria do seu processo pessoal de desenvolvimento.

O PSP provê ao engenheiro de software disciplina no processo de desenvolvimento com a utilização de *scripts*, formulários, modelos e relatórios. Estes recursos do PSP são utilizados na elaboração de processos definidos e estruturados para desenvolvimento, auxiliando na elaboração de estimativas, planos, revisões de projeto e código e outros. A disciplina, no processo de desenvolvimento de software, faz com que sejam produzidos produtos de software de alta qualidade. Assim, atende a requisitos explícitos e implícitos, bem como à entrega do produto no prazo, dentro do custo previamente acordado e com número reduzido de defeitos.

Com a utilização do PSP, o engenheiro de software descobre maneiras de elaboração de estimativas de tempo, de desenvolvimento e organização das tarefas cotidianas. Estas estimativas são possíveis a partir de medições anteriores, efetuadas em tarefas semelhantes. O PSP proporciona, ao engenheiro de software, facilidade para analisar compromissos a serem assumidos, permitindo a decisão de assumir somente aqueles que possam realmente ser cumpridos. Proporciona maneiras de fazer as revisões de projeto e código, na busca e correção de todos os erros possíveis.

A revisão pode ser uma aliada na busca constante por desenvolver produtos de software de alta qualidade. As revisões precisam ser feitas com a utilização de uma lista

de verificação contendo todos os itens que devem ser analisados. A adoção do PSP não é uma tarefa fácil de ser inserida na rotina de trabalho do engenheiro de software. Ela exige mudanças de atitudes, além da análise do processo pessoal de desenvolvimento. Podendo ocasionar barreiras impostas pelos próprios engenheiros e sua utilização. Exige, ainda, a flexibilização na cultura existente de desenvolvimento de software e, isto pode ser um processo de difícil aceitação por parte do engenheiro de software.

Para diminuir o impacto da utilização do PSP, ferramentas de suporte automatizado podem facilitar o trabalho de adoção e aceitação pelos engenheiros de software. A utilização de ferramentas de suporte automatizado pode proporcionar vários benefícios ao engenheiro de software. Um exemplo é a eliminação de erros nos cálculos para análise dos dados colhidos pelo engenheiro de software.

A automação também elimina a cópia e transferência de dados entre formulários, evitando erros durante a transcrição. Além disso, a coleta de dados de tempos e defeitos torna-se simplificada. Outro benefício é a realização automática dos cálculos requeridos. Facilita, assim, inclusive a correção de algumas informações erradas, pois nos processos manuais, vários cálculos precisariam ser realizados novamente.

As informações dos formulários podem ser recuperadas de maneira rápida e precisa. Elimina-se, ainda, a duplicação dos dados nos formulários, o que os torna mais consistentes. Ponto a ser observado na elaboração de uma ferramenta de suporte automatizado, é o atendimento às suas necessidades. Pois, caso esta ferramenta for mal desenvolvida pode tornar-se um empecilho para o engenheiro de software na utilização do PSP.

5.2 PERSPECTIVAS DE TRABALHOS FUTUROS

Como perspectiva para trabalhos futuros propõe-se:

a) implementação desta ferramenta de suporte automatizado para dar suporte a todos os níveis do PSP;
b) implementação de uma ferramenta de suporte automatizado do PSP que interagisse com o compilador ou editor de programa, buscando

informações de maneira automática dos dados de tempo, defeitos, alterações etc, para serem utilizados pela ferramenta sugerida no item a.

5.3 BIBLIOGRAFIA

(ARTHUR, 1994) ARTHUR, L. J. Melhorando a qualidade do software. Rio de Janeiro – RJ, Livraria e Editora Infobooks S.A., 1994.

(COSTA & SANTANA FILHO, 2000) COSTA, A. B., SANTANA FILHO, O. V. Análise da Experiência do Ensino do PSP no SENAC-SP, X CITS, 2000

(CRISTIANO et al., 2002) CRISTIANO, A. et al. Processo Pessoal de Software. Disponível em www.campus.fortunecity.com/princeton/117/psp/psp.htm acesso em 15/07/2002.

(DEMING, 1990) DEMING, W. E. Qualidade: a revolução na administração. São Paulo: Saraiva, 1990

(DISNEY, 1998) DISNEY, A. M. Data quality problems in the Personal Software Process, 1998.

(FERGUSON et al., 1997) FERGUSON, P. et al. Results of Applying the Personal Software Process. IEEE Vol. 30, No. 5; MAY 1997, pp. 24-31

(FOWLER & SCOTT, 2000) FOWLER, M & SCOTT, K. UML Essencial: um breve guia para a linguagem – padrão de modelagem de objetos. 2ª Edição, Porto Alegre, Bookman, 2000.

(FURLAN, 1998) FURLAN, José Davi, Modelagem de objetos através da UML - *the Unified Modeling Language*. São Paulo, Makron Books, 1998.

(GROVE, 1998) Grove, R. F. Using the Personal Software Process to Motivate Good Programming Practices. ACM SIGCSE Bulletin, vol.30, N.3, pp. 98-101, September 1998.

(HAYES & OVER, 1997) HAYES, W. e OVER, J. W. Over The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers. Carnegie Mellon University/SEI. Technical Report CMU/SEI-97-TR-001, 1997.

(HILBURN, 1999) Hilburn T. B. PSP Metrics in Support of Software Engineering Education, 12th Conference on Software Engineering Education and Training, New Orleans, Louisiania

(HOU & TOMAYKO, 1998) HOU, L. e TOMAYKO, J. Applying the Personal Software Process in CS1: an Experiment. ACM SIGCSE Bulletin, Vol.30, N.1, pp. 322-5, March 1998.

(HUMPHREY, 1995) HUMPHREY, W. S. A Discipline for Software Engineering, Addison Wesley, Reading, Mass., 1995.

(HUMPHREY, 1997) HUMPHREY, W. S. Introduction to the Personal Sofware Process, Addison Wesley, Reading, Mass., 1997.

(HUMPHREY, 2000) Humphrey, W. The Personal Software Process. Carnegie Mellon University/SEI. Technical Report CMU/SEI-2000-TR-022, 2000.

(JACOBSON et al, 1992) JACOBSON, I., *Object-Oriented Software Engineering: A Use Case Driven Aproach.* Reading, MA. Addison Wesley, 1992.

(JURAN, 1992) JURAN, J. M. A qualidade desde o projeto: novos passos para o planejamento da qualidade em produtos e serviços. São Paulo: Pioneira, 1992.

(NÉRICI, 1993) NÉRICI, I. G. Educação e Metodologia. Rio de Janeiro: Fundo de Cultura, 1993.

(PAULK et al., 1993) PAULK, M.C. Capability Maturity Model for Software, Version 1.1. Software Engineering Institute Technical Report, CMU/SEI-93-TR1993. Disponível em www.sei.cmu.edu acesso em 20/06/2002.

(PRANTONI, 2002) PRANTONI, G. Estimativa de Tamanho de Software Baseado em Objetos. http://www.choose.com.br/infochoose/artigos/viewer.asp?n=37&a=2
Acesso em 11/08/2002

(PRESSMANN, 1995) PRESSMAN, R. S., Engenharia de Software, São Paulo –

SP, Makron Books, 1995.

(ROCHA et al., 2001) ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. Qualidade de Software: Teoria e Prática, São Paulo, Prentice-Hall, 2001.

(SEI 2002) SEI Software Engineering Institute, Personal Software Process^{sm} (PSP^{sm}) and Team Software Process^{sm} (TSP^{sm}) Results. Disponível em www.sei.cmu.edu/tsp/results.htm acesso em 15/07/2002.

(WEBER et al., 2001) WEBER, K. C., ROCHA, A. R. C., NASCIMENTO, C. J. do. Qualidade e produtividade em software, São Paulo, Makron Books, 2001.

(WILLIAMS, 2000) WILLIAMS, L. A. Adjusting the Instruction of the Personal Software Process to Improve Student Participation.

(ZHONG et al., 2000) ZHONG, X., MADHAJVI N. H., EMAN, K. E. Critical Factors Affecting Personal Software Processes. IEEE Software, Novembro / Dezembro 2000.

(BOOCH et al, 2000) BOOCH, G., RUMBAUGH, J., JACOBSON I. Guia do Usuário UML, São Paulo, Ed. Campus, 2000.