SHEILA DOS SANTOS REINEHR

PSPi - UMA INSTÂNCIA DO *PERSONAL SOFTWARE PROCESS* PARA O AMBIENTE CORPORATIVO

Dissertação apresentada ao Programa de Pós-graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como parte dos requisitos para obtenção do título de Mestre em Ciências.

Área de concentração: Sistemas de Informação

Orientador: Prof. Robert Carlisle Burnett

CURITIBA
2001

Reinehr, Sheila dos Santos

PSPi – Uma instância do *Personal Software Process* para o ambiente corporativo. Curitiba, 2001.

Dissertação (Mestrado) – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. Engenharia de Software 2. Processo de Software 2. Qualidade de Software – I. Pontifícia Universidade Católica do Paraná II. Centro de Ciências Exatas e de Tecnologia III. Programa de Pós-Graduação em Informática Aplicada.

No porto de antes, apreensivo, eu tentava imaginar as dificuldades e lutas futuras. No de agora, dono do tempo que eu conquistara, simplesmente admirava o que estava ao redor e desfrutava do que estava feito. Não era a sensação de uma batalha ganha, de uma luta em que os obstáculos estavam vencidos. Muito mais do que isso, era o prazer interior de ter realizado algo que tanto desejei, de ter feito e visto o que eu fiz e vi.

Amyr Klink

A Deus, que permite à ciência evoluir.

Ao meu marido Vanio, eterno namorado e companheiro de jornada.

À minha filha Giovanna, luz da minha vida.

À minha mãe, Huri, pelas orações e pelas lições de vida.

Ao meu pai, Nelson, que adoraria estar aqui comigo neste momento.

AGRADECIMENTOS

Ao meu marido, Vanio, e à minha filha, Giovanna, que, além de me amar e apoiar, tão pacientemente suportaram as ausências necessárias ao desenvolvimento deste trabalho.

Aos meus pais, Nelson (*in memorian)* e Huri, pela formação, pelos valores cristãos, pelo amor e pelo exemplo de vida.

À minha irmã, Sandra, pelo papel indispensável de tia.

Ao meu orientador, Professor Robert Carlisle Burnett, pela confiança, apoio e inúmeras oportunidades de aprendizado e crescimento.

À analista de sistemas Valderes Maestrelli Zarnicinski, profissional competente e dedicada, pelo apoio durante os trabalhos desta dissertação.

Ao amigo Marco Antonio Paludo que sempre me incentivou a seguir em frente.

À amiga Carla Wanderer, por constantemente me socorrer nas atribuições de mãe.

À amiga Cristina Filipak Machado, companheira de normas e angústias.

À Pontifícia Universidade Católica do Paraná, pela estrutura e suporte.

Aos professores do Mestrado que compartilharam conosco um pouco de seu vasto conhecimento.

Aos colegas de Mestrado, pelas divertidas provas “24 horas”.

Aos amigos da CE21.101.03 – Gerência de Ciclo de Vida - da ABNT pela torcida, carinho e apoio.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CASE	*Computer Assisted Software Engineering*
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integrated* (o produto)
CMMI	*Capability Maturity Model Integration* (o projeto)
CMMI-SE/SW	*CMMI for Systems Engineering/Software Engineering*
CMU	*Carnegie Mellon University*
KPA	*Key Process Area*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
LOC	*Lines of Code* (linhas de código)
PROBE	*PROxy Based Estimating*
PSP	*Personal Software Process*
RUP	*Rational Unified Process*
TSP	*Team Software Process*
USDoD	*United States Department of Defense*
SEI	*Software Engineering Institute*
SW-CMM	*Capability Maturity Model for Software*

RESUMO

A necessidade de expansão das organizações e o avanço da tecnologia têm gerado uma demanda constante por novos produtos e serviços. Porém, as empresas não podem simplesmente descartar as aplicações vigentes, que fornecem apoio aos seus processos de negócio, e substituí-las completamente por novas aplicações. A necessidade de conviver com sistemas de grande porte, em sua maioria sistemas legados e em operação há vários anos, e ao mesmo tempo, disponibilizar produtos e serviços que utilizam todo o potencial das novas tecnologias, é um desafio que faz parte do cotidiano dessas organizações.

Paralelamente, a engenharia de software vem buscando sua maturidade, rumo a tornar-se efetivamente uma disciplina de engenharia. Diversos têm sido os métodos, modelos e normas que buscam trazer para a engenharia de software os mesmos conceitos já conhecidos e praticados na indústria de manufatura: a definição e a melhoria de processos, como forma de reduzir custos e melhorar a qualidade do produto final.

Esta dissertação pretende contribuir com as organizações que possuem ambientes corporativos que não podem ser descartados, a usufruir os benefícios que os novos modelos e normas da engenharia de software têm trazido. Para isso, propõe uma instância do PSP – *Personal Software Process*, modelo que privilegia a melhoria das práticas individuais, como forma de estabelecer o primeiro passo rumo à melhoria organizacional. Essa instância é composta por novos elementos, adicionados ao PSP com o objetivo de melhorar a capacidade de estimativa de tamanho, além de uma ferramenta, PSPPlus-i, que apóia o novo processo.

Como forma de validar a instância PSPi proposta, um estudo de caso é conduzido, utilizando a ferramenta desenvolvida, provendo como um de seus resultados, uma base de categoria de tamanho relativo validada e que pode ser aplicada em organizações com ambiente similar.

ABSTRACT

Due to the increasing need for expansion in all kind of organizations, combined with technologies evolvement, a growing demand for new products and services is being established. But companies cannot simply discard their actual applications, which provide support to their business processes, and completely replace them by new ones. The need to coexist with mainframe systems, major of them legacy systems been operated for several years, and at the same time, deliver products and services using new technologies full potential, is a challenge that takes part of companies daily life.

In the same time, software engineering has been seeking its maturity towards becoming an effective engineering discipline. There are several methods, models and standards aiming to bring to software engineering the same concepts a long time well known in manufacture industry: the definition and the improvement of processes, as a way to reduce costs and improve final product quality.

This dissertation aims at contributing with organizations that have corporative environments, which cannot be discarded, to benefit from new software engineering models and standards. To do so, it proposes an instance of the PSP – Personal Software Process, a model that privileges individual practices improvement, as a means to establish the first step towards an organizational improvement. Such instance is composed by new elements, added to the original PSP in order to improve size estimating capabilities. Besides, a toll, called PSPPlus-I, is developed to provide appropriate support to the new process.

As a way of assessing the proposed instance, PSPi, a study case is conducted using the new tool, providing, as one of its major results, a relative size categories base that can be applied in organizations with similar environments.

CAPÍTULO 1 - INTRODUÇÃO

A engenharia de software é uma disciplina relativamente nova se comparada às demais especialidades conhecidas de engenharia. Por conta disto, os projetos envolvendo software ainda continuam buscando o mesmo rigor dos projetos das engenharias tradicionais. A complexidade e o tamanho dos projetos envolvendo software, quer como componentes isolados ou como parte integrante de um sistema, têm crescido de tal forma que se torna cada vez mais difícil assegurar seu sucesso. Esta dificuldade cresce ainda mais quando se leva em consideração a pressão exercida pelo mercado que exige: ciclos de desenvolvimento mais curtos, melhor qualidade e menor custo. É por este motivo que a comunidade de engenharia de software vem buscando métodos, técnicas, padrões e ferramentas que auxiliem a estabelecer para si o mesmo rigor e disciplina dos outros ramos de engenharia.

Para garantir que projetos de software, cada vez maiores e mais complexos, tenham sucesso, ou seja, possam ser executados com qualidade, dentro do prazo e orçamento, é imprescindível que o processo utilizado na sua produção esteja bem definido e gerenciado. Mark Paulk, em [PAULK et alli, 1994], aborda este assunto da seguinte forma:

> "A premissa que embasa o gerenciamento do processo de software é que a qualidade de um produto de software é em grande parte determinada pela qualidade do processo utilizado para desenvolvê-lo e mantê-lo. Um processo de software efetivo une pessoas, ferramentas e métodos em um todo integrado."

Pressman complementa e estende este conceito em [PRESSMAN, 1997], afirmando que o gerenciamento efetivo do projeto de software depende de três "P"s: *processo, pessoas, e problema*. Visão similar é apresentada por Jacobson em [JACOBSON et alli, 1999] relacionando o desenvolvimento de software a quatro "P"s: *projeto, processo, pessoas e produto.*

A Figura 1 retrata uma adaptação da visão apresentada por [JACOBSON et alli, 1999], complementada com os conceitos de [PRESSMAN, 1997], e ilustra a relação entre os diversos "P"s, cujo significado será melhor explorado nas próximas seções:

- Processo;
- Projeto;
- Pessoas;
- Problema;
- Produto.

Conforme se pode observar na Figura 1, os *projetos* são estabelecidos para solucionar os *problemas* identificados pelos proponentes e/ou patrocinadores. O *processo* funciona como um modelo[1] a partir do qual os *projetos* são instanciados. As *pessoas* participam do *projeto*, desempenhado os mais variados papéis, como mostrado a seguir. E, finalmente, o *produto* é o resultado do *projeto* e, conseqüentemente, uma solução para o *problema*.

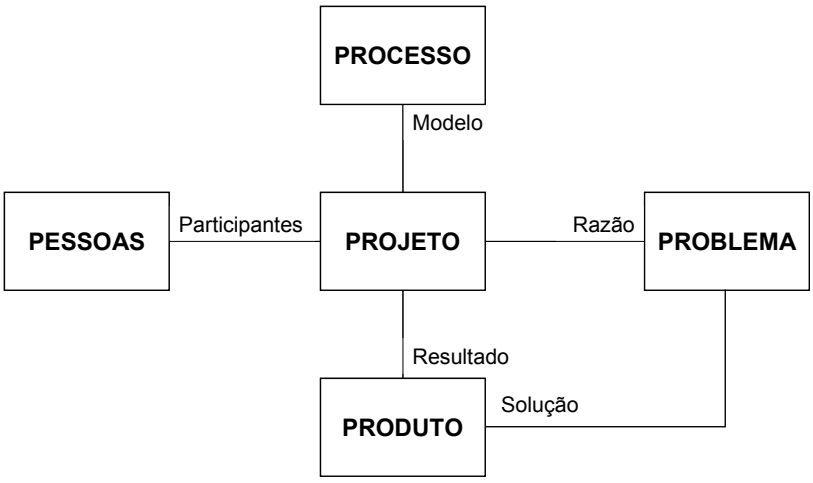

Figura 1. Os 5 "P"s no desenvolvimento de software.

As próximas subseções detalham cada um dos "P"s envolvidos no desenvolvimento de software, sem no entanto, estabelecer qualquer tipo de hierarquia entre eles, pois é da perfeita integração entre todos os "P"s, que resulta o sucesso do projeto de software.

[1] Do original, em inglês, *template*.

1.1 O primeiro "P": o processo

O primeiro "P", *processo,* é definido de forma genérica e abrangente em [ABNT, 1998], como sendo um "conjunto de atividades[2] inter-relacionadas, que transforma entradas em saídas".

Já em [PAULK et alli, 1994] os autores iniciam seu trabalho com a definição de *processo* a seguir:

> "Processo é o que as pessoas fazem, usando procedimentos, métodos, ferramentas e equipamentos, para transformar matéria prima (entradas) em um produto (saída) que é de valor para o cliente."

Ainda, em [PAULK et alli, 1994], Mark Paulk especializa o conceito genérico de processo para o contexto de software e define *processo de software* como sendo:

> "Um processo de software pode ser definido como um conjunto de atividades, métodos, práticas e transformações que as pessoas empregam para desenvolver e manter software e produtos relacionados (como planos de projeto, documentos de projeto, código, casos de teste e manuais de usuário)."

Definição similar de *processo de desenvolvimento de software* é apresentada por Jacobson em [JACOBSON et alli, 1999]:

> "... é a definição do conjunto completo de atividades necessárias para transformar requisitos do usuário em um conjunto consistente de artefatos que representam um produto de software e, posteriormente, para transformar mudanças nestes requisitos em um novo conjunto consistente de artefatos."

No entanto, os autores adotam um conceito um pouco diferente em [FLORAC & CARLETON, 1999], apropriando a visão de Pall[3], que, ao contrário de Paulk, inclui no contexto de processo: as pessoas, os materiais, a energia, os equipamentos e os procedimentos:

> "Um processo pode ser definido como a organização lógica de pessoas, materiais, energia, equipamentos e procedimentos, em atividades de trabalho projetadas para produzir um resultado final especificado."

---

[2] Em nota explicativa, no mesmo documento, é esclarecido que o termo atividades engloba a utilização de recursos.

[3] Os autores utilizam a definição de processo de: Pall, Gabriel A. *Quality Process Management*. Englewood Cliffs, NJ:Prentice Hall, 1987.

O *processo de software,* portanto, é, de forma simplificada, a maneira pela qual entradas (necessidades do usuário[4]) são transformadas em saídas (produto ou serviço de software para o usuário). Melhorar a qualidade do processo utilizado no desenvolvimento e manutenção de software é uma forma de contribuir para a melhoria na qualidade do produto final.

1.2 O segundo "P": o projeto

O segundo "P", *projeto*, pode ser considerado como uma instância do processo de software. O conceito de projeto, de uma forma mais ampla do que apenas o projeto de software, é bem explorado no PMBOK – *Project Management Body of Knowledgement* [PMI, 2000]:

> "Organizações executam trabalhos. Um trabalho geralmente envolve tanto operações como projetos, embora os dois possam se sobrepor. (...) Projetos são normalmente implementados como um meio para atingir o plano estratégico da organização. Operações e projetos diferem basicamente no fato de que operações são contínuas e repetitivas enquanto projetos são temporários e únicos."

Dois conceitos caracterizam, portanto, um projeto: temporário e único. Dizer que um projeto é temporário significa dizer que ele tem um começo definido e um fim definido. Por fim definido pode-se entender, tanto a entrega do produto final, como o próprio cancelamento do projeto, quer seja porque os seus objetivos não poderão ser atingidos, quer seja porque se entende que não há mais necessidade do produto e/ou serviço que ele geraria. Por único entende-se que o produto e/ou serviço que é gerado pelo projeto é, sob algum aspecto, diferente de outros produtos ou serviços, mesmo que pertencente à mesma categoria.

Projeto pode então ser definido como [PMI, 2000]:

> "Um projeto é um empreendimento temporário iniciado para criar um produto ou serviço único."

Gerenciar um projeto, qualquer que seja a sua natureza, significa aplicar de forma adequada conhecimento, perfis, ferramentas e técnicas para atingir seus requisitos [PMI, 2000]. Quanto mais definido e mensurável for o processo a partir do qual o

[4] Usuário, em [ABNT, 1998], é definido como: "indivíduo ou organização que utiliza um sistema em operação para executar uma função específica".

projeto é instanciado, mais facilmente ele poderá ser gerenciado, ou seja, mais fácil será manter o equilíbrio necessário entre os recursos acima citados.

1.3 O terceiro "P": as pessoas

O terceiro "P", *pessoas,* vem da constatação de que o conhecimento é a matéria prima para se desenvolver software e são as *pessoas* que transformam conhecimento em produtos de software [CURTIS et alli, 1995a]. Pessoas capacitadas, treinadas e motivadas constituem um dos pilares que fundamentam o conceito de processo de software.

Em 1988, Curtis [CURTIS et alli, 1988] já citava as afirmações de dois vice-presidentes de grandes empresas de software:

> "O ingrediente mais importante neste projeto de sucesso foi ter pessoas inteligentes... Muito pouca coisa importa na minha opinião... A coisa mais importante que você faz por um projeto é selecionar a equipe... realmente o sucesso de organizações que desenvolvem software é muito, muito associado à sua habilidade de recrutar pessoas boas."

> "A única regra que eu tenho em gerenciamento é assegurar que eu tenho pessoas competentes – pessoas realmente competentes – e que eu promovo o crescimento das pessoas competentes, e que eu forneço um ambiente onde essas pessoas possam produzir."

Em [JACOBSON et alli, 1999], os autores também ressaltam a importância das pessoas no processo de software, porém o fazem de forma mais abrangente, incluindo não somente os desenvolvedores em si, mas todos aqueles que direta ou indiretamente dão forma ao produto final:

> "Pessoas estão envolvidas no desenvolvimento de um produto de software durante todo o seu ciclo de vida. Elas financiam o produto, fazem o cronograma, desenvolvem, gerenciam, testam, usam e se beneficiam dele. Portanto, o processo que guia este desenvolvimento deve ser orientado a pessoas, ou seja, aquele que funciona bem para as pessoas que o utilizam."

Em [GIBSON, 1997], o autor afirma que à medida que as organizações se aproximam do nível 3 do modelo SW-CMM[5], descobrem que níveis superiores de maturidade são dependentes da melhoria do desempenho individual. Isso significa que

[5] O modelo SW-CMM – *Capability Maturity Model for Software* - é apresentado em detalhes no Capítulo 2 desta dissertação.

não é suficiente que os processos organizacionais estejam claramente definidos e descritos, pois, em última instância, são pessoas que dão forma ao produto final.

Ainda no mesmo artigo, o autor coloca que:

> "Independentemente de quão bem uma organização é gerenciada, a qualidade do trabalho de software é governada pelas práticas dos engenheiros de software."

Isso não significa, no entanto, um detrimento da importância do gerenciamento, em função da importância das práticas individuais. Significa, sim, que se deve dedicar particular atenção às práticas individuais, no momento de gerenciar os projetos de software.

Do mesmo modo, isso não significa que se deva alienar a responsabilidade pela qualidade do produto de software apenas aos engenheiros de software. Qualidade não é adicionada ao produto por algumas pessoas. Ao contrário, qualidade é de responsabilidade de cada membro da organização desenvolvedora [KRUCHTEN, 2000].

E, finalmente, são pessoas que desenvolvem software, e o fazem utilizando um processo, que tanto pode ser formal, como não. Processos informais são decorrentes de práticas incorporadas ao cotidiano dos desenvolvedores de forma não planejada, definida ou estruturada. Quanto mais formal for este processo, ou seja, quanto mais bem definido, mensurável e visível for o processo, mais previsíveis serão o seu desempenho e sua qualidade.

1.4 O quarto "P": o problema

O quarto "P", o *problema*, é a razão de existir de todo o projeto de software. É para solucionar um problema identificado no universo do usuário que os projetos de software são iniciados. Podemos entender problema, nesse contexto, de uma forma bem ampla, que engloba, tanto um problema propriamente dito, como uma oportunidade de evolução. Logo, os projetos iniciados podem tanto se referir a uma manutenção preventiva, corretiva ou evolutiva em produtos ou serviços de software existentes, quanto ao desenvolvimento de um novo produto ou serviço de software.

A compreensão do problema faz parte das fases inicias do ciclo de vida do software, mais especificamente das fases de elicitação e análise de requisitos, tratadas

na engenharia de software pela área conhecida como engenharia de requisitos. É imprescindível que o problema esteja bem compreendido para que o resultado final seja um produto de qualidade e que agregue valor para o cliente.

Pressman aborda essa questão de forma bastante apropriada em [PRESSMAN, 1997], quando relaciona a questão das pessoas e do problema:

> "Um gerente que falha em encorajar uma comunicação abrangente com o cliente logo no início do projeto, corre o risco de construir uma solução elegante para o problema errado."

Diversas definições para engenharia de requisitos foram compiladas em [ZANLORENCI & BURNETT, 1999]. Dentre elas, encontra-se a de [SOMMERVILLE & SAWYER, 1997], que define a engenharia de requisitos como sendo:

> "... um termo relativamente novo que foi inventado para cobrir todas as atividades envolvidas em descobrir, documentar e manter um conjunto de requisitos para um sistema baseado em computador. (...) O termo 'engenharia de requisitos' vem do campo da engenharia de sistemas; para os que vêm do campo de sistemas comerciais, podem pensar em engenharia de requisitos mais ou menos como análise de sistemas."

A importância da etapa relativa aos requisitos é ressaltada em diversos pontos da literatura de engenharia de software e de requisitos. Frederick Brooks, em seu clássico *No Silver Bullet*[6]*,* [BROOKS, 1987], aponta esta etapa como uma das mais críticas para o sucesso do projeto de software:

> "A parte mais difícil de construir um sistema de software é decidir precisamente o que construir. Nenhuma outra parte do trabalho conceitual é tão difícil quanto estabelecer os requisitos técnicos detalhados, incluindo todas as interfaces com as pessoas, as máquinas e outros sistemas de software. Nenhuma outra parte do trabalho danifica tanto o sistema resultante se for feita de forma errada. Nenhuma outra parte é mais difícil de retificar posteriormente."

Requisitos mal compreendidos e mal definidos são uma causa bastante freqüente do insucesso de projetos de software. Muitos projetos, apesar de concluídos e implantados, nunca chegam a ser usados, ou o são de forma parcial. Em algum nível não se tornaram aderentes às expectativas de quem os definiu, inviabilizando sua utilização.

---

[6] O artigo em questão foi reeditado na edição de aniversário do livro *The Mythical Man-Month.*[BROOKS, 1995]

1.5 O quinto "P": o produto

Finalmente, o quinto e último, porém não menos importante "P", o *produto*. É através do produto que o problema é solucionado pelo projeto de software, ou seja, o produto é a finalidade do projeto e, em última instância, do processo de software.

É importante explorar um pouco mais o conceito de produto encontrado na literatura, para não limitá-lo a um escopo excessivamente restritivo.

Jacobson apresenta em [JACOBSON et alli, 1999], um conceito bastante abrangente e adequado, iniciando o texto com a afirmação de que produto de software é mais do que código:

> "(...) produto refere-se não somente ao código que é entregue, mas ao sistema como um todo. (...) Um sistema é o conjunto completo dos artefatos que o representam de forma que ele possa ser compreendido pela máquina ou pelas pessoas, para: as máquinas, os participantes[7] e as partes interessadas."

Ainda no mesmo texto, o autor explicita o que entende por cada uma dessas partes:

> "As máquinas são: ferramentas, compiladores e computadores. Participantes incluem: gerentes, arquitetos, desenvolvedores, testadores, profissionais de marketing, administradores, e outros. Partes interessadas são autoridades de recursos, usuários, pessoal de vendas, gerentes de projeto, gerentes de linha, pessoal de produção, órgãos regulamentadores, e assim por diante."

Ao entender que diversas visões devem ser satisfeitas através do produto de software, o autor engloba outros artefatos produzidos durante a concepção e o desenvolvimento de software, além de apenas o código em si. Essa é uma abordagem muito importante para que se compreenda um pouco mais sobre qualidade de produto.

Assim como qualidade não é de responsabilidade de apenas uma pessoa, mas sim de toda a organização [KRUCHTEN, 2000], qualidade tão pouco é adicionada ao produto de software em sua fase final. Qualidade é adicionada em cada uma das etapas do ciclo de desenvolvimento e precisa ser assegurada em cada um dos artefatos (produtos) intermediários produzidos.

---

[7] O original, em inglês, utiliza o termo *workers* no sentido de expressar os diversos papéis que um participante do projeto de software pode assumir. Optou-se pela palavra participante, mesmo sabendo que não traduz literalmente o termo *worker*.

1.6 O cenário atual

No primeiro capítulo de [KRUCHTEN, 2000], Grady Booch desenha, de forma bastante adequada, o cenário atual da indústria de software:

> "A boa notícia para os profissionais de software é que a economia mundial depende cada vez mais de software. Os sistemas que utilizam software intensivamente, que a tecnologia torna possível e a sociedade demanda, estão aumentando em tamanho, complexidade, distribuição e importância. A notícia ruim é que a expansão desses sistemas em tamanho, complexidade, distribuição e importância, empurra os limites do que nós, na indústria de software, sabemos como desenvolver. Tentar evoluir sistemas legados para tecnologias mais modernas traz o seu próprio conjunto de problemas técnicos e organizacionais. Para aumentar o problema, o negócio continua a demandar produtividade crescente e melhor qualidade, com desenvolvimento e entrega mais rápidos. Adicionalmente, ainda, a disponibilidade de pessoal de desenvolvimento qualificado não é condizente com a demanda."

O autor conclui seu texto com a seguinte afirmação:

> "O resultado é que construir e manter software é difícil e está ficando mais difícil; construir software de qualidade de forma repetível e previsível é mais difícil ainda."

No contexto desta dissertação, optou-se por considerar para a expressão sistema legado, a definição apresentada no glossário geral de [JACOBSON et alli, 1999]:

> "Sistema legado: um sistema existente herdado por um projeto. Usualmente um sistema velho criado usando tecnologias de implementação mais ou menos obsoletas, mas que precisa ser incorporado ou reusado – quer em sua totalidade, quer parcialmente – quando um novo sistema é construído pelo projeto."

Observa-se, então que se tem: por um lado, sistemas legados, que são a base da organização e que precisam ser mantidos e evoluídos; por outro, uma oferta crescente de novas tecnologias e uma demanda, também crescente, do mercado consumidor pelos produtos e serviços viabilizados por essas novas tecnologias. Como então manter e incrementar o valor dos sistemas legados como um ativo da organização, ao invés de depreciá-lo ao longo do tempo como um imobilizado?

Migrar os sistemas legados para novas tecnologias pode ser uma das alternativas. Porém, tanto a migração completa, como a modernização parcial, não são tarefas triviais e têm sido objeto de investigação constante, conforme evidenciado em [SEACORD et alli, 2001] onde os autores discutem estratégias de modernização de sistemas legados:

> "A modernização de sistemas corporativos legados introduz muitos desafios relativos ao tamanho, complexidade e fragilidade dos sistemas legados. Os aspectos de tamanho e complexidade freqüentemente ditam que esses sistemas sejam modernizados incrementalmente, e novas funcionalidades sejam incrementalmente implantadas antes do esforço de modernização estar concluído. Isso, por sua vez, requer que os componentes legados operem lado a lado com os componentes modernizados em um sistema de operação que introduz novos problemas."

É preciso, portanto, mesmo que se opte por uma estratégia de migração, conviver com os sistemas legados e continuar evoluindo-os até que a migração esteja concluída. É possível, até mesmo que, por uma questão de custo x benefício, se opte por não efetuar a migração e continuar mantendo apenas os sistemas legados.

Como então garantir que os objetivos estratégicos da organização continuem a ser atingidos, diante desse cenário de incertezas? Além disso, como garantir que o sejam com os recursos disponíveis?

Não há respostas prontas, mas uma crescente preocupação da comunidade de engenharia de software em prover uma solução para essas questões, detendo-se com mais atenção sobre cada um dos fatores descritos nas seções anteriores através dos cinco "P"s.

Por perceber que o produto é mais do que o código entregue, e que a qualidade desse produto é influenciada pela qualidade do processo utilizado em sua produção, o foco da comunidade de tecnologia da informação voltou-se para os modelos de melhoria de processo. Pode-se notar que essa não constitui uma visão inovadora, mas, sim, uma visão tomada por empréstimo da indústria de manufatura. Ao se compreender que o processo é o elo chave que liga todos os demais aspectos abordados, diversos modelos de melhoria de processos de software passaram a ser desenvolvidos, utilizados e evoluídos[8].

A preocupação com essas questões nas empresas de software do Brasil pode ser claramente observada na pesquisa realizada bienalmente pelo Ministério da Ciência e Tecnologia sobre Qualidade e Produtividade no Setor de Software Brasileiro, a qual aponta para índices inicialmente tímidos, porém sempre crescentes, de conhecimento e aplicação de normas, modelos, métricas e práticas de engenharia de software de uma forma geral [MCT, 2000].

[8] Esses modelos são extensivamente discutidos no capítulo 2 desse trabalho.

É sabido, no entanto, que a aplicação e utilização extensiva das práticas que representam o estado da arte em engenharia de software nem sempre é fácil ou condizente com a realidade das empresas, especialmente as nacionais [MCT, 2000]. Deslocar recursos da linha de frente de desenvolvimento, sempre escassos, para compreender, medir, definir e alterar processos internos de desenvolvimento de software, nem sempre é visto como uma prioridade. Embora as corporações tenham a plena consciência de que estão vivenciando o dilema colocado por Booch, é muito difícil quebrar o círculo vicioso.

Integrar esses cinco aspectos que envolvem a questão do desenvolvimento de software, e evoluí-los para abranger, tanto o sistema legado, quanto os novos desenvolvimentos, geralmente não é uma tarefa trivial, pois exige uma multi-disciplinaridade que nem sempre está disponível nas organizações. Compreender e instanciar os modelos e normas genericamente concebidos para que possam endereçar adequadamente essas questões no contexto da organização tão pouco são tarefas triviais.

1.7 Motivação

Levando-se em consideração o cenário atual descrito na seção anterior, que confirma as palavras de Booch em [KRUCHTEN, 2000], que à medida que as empresas crescem e a tecnologia se torna disponível, cresce também a necessidade e a demanda por novos produtos e serviços; e o fato de que não é possível simplesmente desprezar todos os sistemas legados corporativos [SEACORD et alli, 2001], optou-se, nesta dissertação, por desenvolver uma forma de evoluir os processos desse ambiente, tornando-os aderentes às práticas atuais de engenharia de software.

Partindo da afirmação de [GIBSON, 1997] que não basta ter processos organizacionais eficientes, pois, em última instância, são as pessoas que desenvolvem os produtos de software, optou-se, nesta dissertação, por tomar como base um modelo que privilegiasse o desempenho individual, sem contudo negligenciar os demais “P”s abordados anteriormente. Para isso, será utilizado como ponto de partida o modelo destinado a indivíduos e pequenas equipes de projeto, o PSP – *Personal Software*

Process [HUMPHREY, 1995], que pode ser considerado como o primeiro passo rumo à melhoria organizacional[9].

O PSP, no entanto, conforme se verá nos próximos capítulos, não pode ser usado diretamente pela organização sem que adaptações para aderência ao contexto tenham que ser efetuadas. Portanto, como forma de auxiliar as empresas que convivem com sistemas legados corporativos, e os desenvolvedores dessas empresas, a iniciarem a utilização do PSP, a presente dissertação se propõe a desenvolver uma instância do modelo para esse ambiente, que será denominada PSPi. Além disso, apresenta uma ferramenta para apoiar o processo proposto, que será denominada PSPPlus-i.

A instância PSPi, apresentada por essa dissertação, se propõe a[10]:

- Instanciar o processo de software destinado a pessoas, PSP – *Personal Software Process* [HUMPHREY, 1995] para ser utilizado em projetos de ambientes corporativos legados de grande porte;
- Montar e testar uma base de estimativas de tamanho de programas para o ambiente legado de grande porte;
- Prover orientações que reduzam o aspecto subjetivo do método de estimativas PROBE utilizado pelo PSP;
- Desenvolver uma ferramenta que automatize a montagem da base de estimativas;
- Exercitar essa instância do PSP, bem como a base, as orientações e a ferramenta, no ambiente corporativo de uma empresa de grande porte, validando seus resultados.

Com essa abordagem se pretende oferecer às empresas que possuem um ambiente legado corporativo expressivo, e que não pode ser desprezado, uma forma de iniciar o seu programa de melhoria de processos. Não se pretende, com este trabalho, substituir os métodos já consagrados, mas sim, combiná-los e estendê-los, de forma a oferecer uma visão integrada, abrangente e pragmática.

[9] Os motivos para a escolha do PSP são explorados com mais detalhe ao final do capítulo 2, após a apresentação dos modelos voltados para processo de software.
[10] Alguns termos são típicos do PSP e serão esclarecidos no decorrer deste trabalho, bem como o próprio modelo PSP, descrito em detalhes no capítulo 3.

1.8 Estrutura deste trabalho

De forma a facilitar a leitura e compreensão desta dissertação, seus tópicos estão assim distribuídos:

- O capítulo 2 apresenta e analisa os modelos de melhoria de processos de software existentes atualmente, incluindo as normas internacionais voltadas para os processos de software;
- O capítulo 3 mostra, de forma mais detalhada, cada um dos níveis de maturidade do PSP e analisa os seus pontos fortes e as oportunidades de melhoria. Além disto, faz referência aos trabalhos encontrados sobre PSP no Brasil e no exterior, tanto na academia, quanto na indústria;
- O capítulo 4 introduz a instância PSPi que é a proposta desta dissertação;
- O capítulo 5 descreve a ferramenta PSPPlus-i desenvolvida por esta dissertação para apoiar a instância PSPi;
- O capítulo 6 apresenta o estudo de caso desenvolvido utilizando tanto a instância PSPi proposta, como a ferramenta PSPPlus-i desenvolvida;
- O capítulo 7 conclui este trabalho e aponta trabalhos futuros que podem dele ser derivados.

1.9 Considerações sobre o capítulo

O presente capítulo conceituou e analisou a importância dos 5 “P”s no desenvolvimento de software: processo, projeto, pessoas, problema e produto. Além disso, apresentou o cenário atual da indústria de software. Com essas informações, conceituou e delimitou o trabalho desenvolvido na presente dissertação, estabelecendo o seu foco no ambiente legado corporativo e na instanciação do modelo de melhoria individual PSP. Ao final, apresentou a estrutura que será utilizada para a abordagem dos tópicos.

CAPÍTULO 2 - MODELOS DE PROCESSO

Partindo-se do princípio que a qualidade do produto de software é grandemente influenciada pelo processo utilizado para produzi-lo [PAULK et alli, 1994], diversos modelos que avaliam e auxiliam na melhoria dos processos de software foram desenvolvidos e vêm sendo utilizados na indústria. Alguns são ditos aplicáveis em nível organizacional, outros, em nível pessoal ou de pequenos times.

Além desses modelos, há ainda esforços internacionais no sentido de padronização para os processos envolvendo o ciclo de vida de software e avaliação da capacidade desses processos.

Esses modelos e padrões serão apresentados em três etapas:

- Modelos de qualidade de processo de software em nível organizacional;
- Modelos de qualidade de processo de software em nível pessoal ou de pequenos times;
- Esforços de padronização na área de processos de software.

2.1 Modelos de Qualidade de Processo de *Software* em nível organizacional

O *Software Engineering Institute* (SEI), da *Carnegie Mellon University* (CMU), sob o patrocínio do *United States Department of Defense* (US-DoD), desenvolveu um conjunto de modelos para guiar as empresas na melhoria da qualidade dos projetos envolvendo software. O princípio adotado pelo SEI na construção destes modelos é que, conforme citado anteriormente, a qualidade do produto de software está intimamente ligada à qualidade do processo utilizado na produção deste produto.

O que estes modelos têm em comum é o fato de que são baseados em níveis de maturidade da capacidade do processo. Cada nível de maturidade contém uma série de práticas que devem ser executadas pela organização para migração para o nível seguinte. À medida que a organização caminha em direção aos níveis superiores de maturidade, aumentam sua capacidade de previsão, sua efetividade e seu controle sobre os projetos de software.

Os modelos atualmente sendo desenvolvidos e/ou melhorados pelo SEI e relacionados com os processos organizacionais são:

- *Capability Maturity Model for Software* (SW-CMM)
- *Systems Engineering Capability Maturity Model* (SE-CMM)
- *Capability Maturity Model-Integrated* (CMMI)
- *Software Acquisition Capability Maturity Model* (SA-CMM)
- *People Capability Maturity Model* (P-CMM)

Além dos modelos de maturidade propostos pelo SEI, há outros que deles derivaram, como o modelo *Trillium*, desenvolvido pela Bell Canadá [CANADA, 1992].

2.1.1 Capability Maturity Model for *Software* (SW-CMM)

O *Capability Maturity Model for Software* (SW-CMM) teve o seu desenvolvimento iniciado pelo SEI, em 1986, com o patrocínio do US-DoD. O objetivo inicial era o de auxiliar o governo americano a selecionar e gerenciar subcontratos de software, através de um questionário que avaliava o nível de maturidade dos processos da organização. A primeira versão do SW-CMM foi liberada para o público em 1991 e a versão seguinte (v1.1) foi apresentada em forma de relatório técnico publicado pelo SEI em 1993 [PAULK et alli, 1993]. No ano seguinte, esta versão foi complementada com o livro contendo orientações para sua implementação [PAULK et alli, 1994]. Desde então, o SW-CMM, ou simplesmente CMM como é conhecido, vem sendo usado por muitas empresas em diversos países como uma forma de definir, avaliar e melhorar seus processos de *software*. A versão mais atual do modelo faz parte de um conjunto de produtos denominado CMM-I *(Capability Maturity Model Integrated*) descrito no item 2.1.3.

O SW-CMM está estruturado em 5 níveis evolutivos de maturidade, conforme ilustrado na Tabela 1. Níveis mais baixos de maturidade indicam empresas com pouca capacidade de previsão, pouca capacidade de gerenciamento de seus projetos de software, custos elevados de manutenção, baixa produtividade e riscos na adoção de novas tecnologias. Já organizações com níveis mais elevados de maturidade, não apenas conseguem maior previsibilidade sobre suas atividades, como realizam gerenciamento qualitativo e quantitativo de seus projetos, apresentam maior índice de sucesso, menor custo de desenvolvimento e menos riscos ao adotar uma nova tecnologia.

Exceto pelo nível 1, a cada um dos demais níveis estão associadas *Key Process Areas*, ou simplesmente KPAs como são conhecidas, que são os processos e práticas associadas que a empresa deve focar para satisfazer os objetivos daquele nível, conforme ilustrado também Tabela 1. Uma empresa só atinge níveis superiores de maturidade após ter cumprido todas as exigências dos níveis anteriores. Portanto, uma organização, para atingir o nível 3, deve ter implementado, além das KPAs do nível 3, também aquelas previstas para o nível 2.

NÍVEL	DESCRIÇÃO	KPAS
Nível 5	**Otimizado**	Prevenção de Defeitos Gerenciamento da Mudança Tecnológica Gerenciamento da Mudança de Processo
Nível 4	**Gerenciado**	Gerenciamento Quantitativo do Processo Gerenciamento da Qualidade de Software
Nível 3	**Definido**	Foco no Processo da Organização Definição do Processo da Organização Programa de Treinamento Gerenciamento Integrado de Software Engenharia de Produto de Software Coordenação Intergrupos Revisão por Pares
Nível 2	**Repetível**	Gerenciamento de Requisitos Planejamento do Projeto de Software Acompanhamento e Supervisão de Projeto Gerenciamento de Subcontratos de Software Garantia da Qualidade de Software Gerenciamento de Configuração de Software
Nível 1	**Inicial**	-

Tabela 1. SW-CMM - Capability Maturity Model for Software [PAULK et alli, 1993].

Resumidamente, os níveis evolutivos do SW-CMM podem ser assim descritos:

- **Nível 1 – Inicial:** Neste estágio, as empresas podem até desenvolver projetos de sucesso. Porém o fazem com base no heroísmo individual de seus técnicos. Como há sobrecarga de tarefas, em momentos de crise, as fases de planejamento e análise são substituídas por codificação e teste. O processo de desenvolvimento pode ser tão desorganizado que chega a ser caótico, não fornecendo visibilidade ao meio externo. Recursos diversos entram e produtos de software eventualmente saem.

- **Nível 2 – Repetível:** Um gerenciamento mínimo de projeto é estabelecido e já é possível acompanhar custos, cronogramas e funcionalidades. Também já é possível repetir o sucesso de projetos anteriores, desde que sejam similares. Melhor visibilidade do projeto é oferecida ao meio externo, porém ainda não existe um padrão organizacional de processo.
- **Nível 3 – Definido:** Neste momento a empresa já estabeleceu padrões para o desenvolvimento e o gerenciamento de projetos em níveis organizacionais. Todos os projetos utilizam uma instância aprovada do processo padrão da empresa para o desenvolvimento e manutenção de software. A comunidade de usuários já tem grande visibilidade sobre o andamento do projeto.
- **Nível 4 – Gerenciado:** O processo de software é gerenciado quantitativamente. Os riscos podem ser gerenciados de forma mais precisa porque são baseados em métricas colhidas ao longo de todos os projetos. A produtividade e a qualidade são medidas, o que oferece uma grande visibilidade do processo para o meio externo.
- **Nível 5 – Otimizado:** As melhores práticas de engenharia de software são absorvidas e disseminadas por toda a organização. Há um processo contínuo e organizado de mudança. O impacto da introdução de novas tecnologias é minimizado. A comunidade de usuários e a empresa trabalham em forte parceria.

2.1.2 Systems Engineering Capability Maturity Model (SE-CMM)

O *Systems Engineering Capability Maturity Model* (SE-CMM) é o resultado de um trabalho conjunto de indivíduos provenientes do SEI, da academia, do governo americano e da indústria. Ele veio em resposta aos anseios da indústria no sentido de um modelo que auxiliasse a melhoria dos processos de engenharia de sistemas. O SE-CMM identifica elementos essenciais que devem estar presentes nas organizações para o bom uso da engenharia de sistemas. Além disto, provê parâmetros para uma auto-avaliação comparando as práticas vigentes na empresa e as práticas de engenharia de sistemas.

Devido ao fato de que o SE-CMM não aborda especificamente os processos relacionados aos produtos de software, mas sim processos de engenharia de sistemas, não entraremos em detalhes sobre este modelo. Referências podem ser encontradas em [BATE et alli, 1995].

2.1.3 Capability Maturity Model-Integrated (CMMI)

O *Capability Maturity Model-Integrated* (CMMI) é o resultado de um projeto[11] estabelecido em fevereiro de 1998, do qual participaram: o governo americano, a indústria e o SEI. O objetivo era o desenvolvimento de um *framework* integrando os diversos modelos de maturidade (CMMs), de modo a facilitar sua utilização conjunta, preservando os investimentos efetuados na melhoria de processos, tanto da indústria, quanto do governo. Além disto, o projeto buscou aproximar o CMM das definições que estão sendo padronizadas através da futura norma ISO/IEC 15504, discutida posteriormente no item 2.3.3.

O conjunto de produtos resultante deste projeto, CMMI, pode ser utilizado para engenharia de software (CMMI-SW), engenharia de sistemas (CMMI-SE), engenharia de software e sistemas conjuntamente (CMMI-SW/SE) e desenvolvimento integrado de produtos utilizando *Integrated Product and Process Development* (IPPD)[12]. Os *frameworks* CMMI-SW e CMMI-SE não existirão isoladamente, mas sim poderão ser derivados do modelo mais genérico, CMMI-SW/SE. Da mesma forma, o CMMI-IPPD também fará parte de um *framework* mais genérico denominado CMMI-SW/SE/IPPD, não existindo isoladamente como um único modelo.

A primeira versão do CMMI-SW/SE (V0.1) foi enviada aos patrocinadores do projeto para revisão no final de 1998. Este processo envolveu aproximadamente 50 empresas e gerou cerca de 4000 comentários. As principais críticas eram de que o modelo era grande e complexo, não distinguia aspectos normativos de aspectos informativos, tinha pouca integração entre os elementos do modelo e alguma confusão entre áreas de processo da engenharia de sistemas e da engenharia de software.

---

[11] Uma confusão comum existe no que diz respeito à nomenclatura CMMI: o projeto foi denominado *Capability Maturity Model Integration* e o produto resultante foi denominado *Capability Maturity Model Integrated*.

[12] O IPD-CMM atualmente não existe como um produto isolado, mas apenas no CMMI. Originalmente o IPD-CMM foi concebido como um conjunto de disciplinas que devem ser aplicadas ao projeto, desenvolvimento, avaliação e melhoria de desenvolvimento integrado de produto.

No segundo semestre de 1999 o CMMI evoluiu para a versão V0.2, incorporando as modificações necessárias para atender às sugestões recebidas dos patrocinadores. Desta vez o material [SEI, 1999a] foi disponibilizado para avaliação e comentários da comunidade. Em novembro de 2000, após avaliar e incorporar mais de 3000 comentários recebidos da comunidade, a versão atual foi disponibilizada (V1.02).

Uma das alterações mais significativas que o modelo sofreu em relação ao SW-CMM foi passar a oferecer duas formas de representação:

- Representação em estágios [SEI, 1999b], [SEI, 2000a]: apresenta o modelo em termos de níveis de maturidade organizacional e as suas correspondentes áreas de processo[13] (de modo a manter o padrão com o SW-CMM);
- Representação contínua [SEI, 1999c], [SEI, 2000b]: apresenta o modelo de forma contínua, ou seja, sem agrupar as áreas de processo em níveis de maturidade (de modo a aproximar-se da ISO/IEC TR 15504).

Ao utilizar estas duas formas de representação, o modelo passa a conviver com dois conceitos distintos: a maturidade (organizacional) e a capacidade (dos processos).

A maturidade organizacional continua sendo avaliada em níveis de 1 a 5, como no SW-CMM, exigindo que a empresa cumpra todos os requisitos do nível e de seus anteriores para que possa ser certificada nesse nível.

A capacidade dos processos permite a avaliação de determinado processo em níveis de capacidade que vão de 0 a 5, como na ISO/IEC TR 15504. Isso permite que a organização foque a melhoria nos processos que julga mais importantes para o seu contexto.

A Tabela 2 ilustra as áreas de processo do CMMI agrupadas de acordo com as categorias de processos provenientes do relatório técnico ISO/IEC TR 15504:

- Gerenciamento de Processos;
- Gerenciamento de Projetos;
- Engenharia; e
- Apoio.

[13] No CMM-I as KPAs (*Key Process Áreas)* passaram a se chamar simplesmente áreas de processo (*Process Áreas).*

O fato de estarem assim categorizadas não significa que cada uma das práticas esteja restrita a uma determinada área de atuação, pois as áreas de processo possuem grande interação e interdependência umas em relação às outras.

GERENCIAMENTO DE PROCESSOS • Foco no Processo Organizacional • Definição do Processo Organizacional • Treinamento Organizacional • Desempenho do Processo Organizacional • Distribuição das Inovações dos Processos	**GERECIAMENTO DE PROJETOS** • Planejamento de Projeto • Controle e Acompanhamento de Projeto • Gerenciamento de Contrato com Fornecedores • Gerenciamento Integrado de Projeto • Gerenciamento de Risco • Gerenciamento Quantitativo de Projeto
ENGENHARIA • Definição dos Requisitos • Gerenciamento dos Requisitos • Solução Técnica • Integração do Produto • Verificação • Validação	**APOIO** • Gerenciamento de Configuração • Garantia da Qualidade de Processo e de Produto • Medição e Análise • Análise de Decisão e Resolução • Análise Causal e Resolução

Tabela 2. Áreas de Processo do CMMI categorizadas [SEI, 2000a] .

Conforme citado anteriormente, os esforços do projeto CMMI não eram apenas na direção de integrar a engenharia de sistemas e a engenharia de software, mas no sentido de abranger também o desenvolvimento integrado de produto e processo (IPPD). Desta forma, além do modelo CMMI-SE/SW também já está disponível para o público a versão V1.02 do modelo CMMI – SE/SW/IPPD, nas representações: contínua [SEI, 2000c] e em estágios [SEI, 2000d].

Nova versão dos modelos da família CMM-I serão disponibilizados para o público em dezembro/2001.

2.1.4 Software Acquisition Capability Maturity Model (SA-CMM)

O *Software Acquisition Capability Maturity Model* (SA-CMM) é um *framework* para auxiliar a promover melhorias no processo de aquisição de software das empresas. É o resultado de um esforço conjunto entre o SEI, o governo americano e a indústria.

O modelo segue a mesma estrutura básica da família CMM, porém com ênfase nas atividades que dizem respeito ao processo de aquisição de produtos de software. Enquanto o SW-CMM foca os processos do desenvolvedor de software, o SA-CMM

foca os processos do adquirente de software. Por não ser o foco desta dissertação, não o detalharemos, porém a descrição completa poderá ser encontrada em [COOPER et alli, 1999].

2.1.5 People Capability Maturity Model (P-CMM)

O *People Capability Maturity Model*, ou simplesmente P-CMM, é um modelo de maturidade que também segue os princípios da família CMM. Foi desenvolvido pelo SEI e é descrito em [CURTIS et alli, 1995a] como sendo:

> "... um *framework* de maturidade que provê diretrizes para melhorar continuamente a habilidade das organizações de *software* de: atrair, desenvolver, motivar, organizar, e manter, de forma estável, os talentos necessários para melhorar sua capacidade de desenvolver *software*."

Sua primeira versão (1.0) foi liberada para o público em 1995 [CURTIS et alli, 1995a] e aplicada com sucesso em empresas como: IBM, Boeing, BAESystems, Tata Consultancy Services, Ericsson, Lockheed Martin, QAI (Índia), entre outras. Sua segunda versão (2.0) contou com a colaboração e a revisão de empresas e profissionais individuais, tendo sido liberada para o público no segundo semestre de 2001 [CURTIS et alli, 2001].

O foco principal do P-CMM está nas pessoas envolvidas no desenvolvimento e manutenção de software. Assim como os demais modelos da família CMM, o P-CMM também está estruturado em 5 níveis de maturidade e suas correspondentes KPAs, completamente descritas em [CURTIS et alli, 1995b] e [CURTIS et alli, 2001].

As KPAs do P-CMM estão organizadas em 4 trilhas, ou áreas de conhecimento, conforme ilustra a Tabela 3:

- Desenvolvendo capacidades individuais;
- Construindo equipes de trabalho e cultura;
- Motivando e gerenciando o desempenho;
- Desenvolvendo recursos.

Devido ao fato de que o P-CMM é um modelo relativamente novo, liberado em 1995, os dados de utilização ainda são modestos e pouco divulgados. De acordo com os autores em [CURTIS et alli, 2001], 40% das empresas certificadas em níveis 4 e 5, segundo o modelo SW-CMM, estão utilizando o P-CMM para promover melhorias em

seus processos relativos à gestão e desenvolvimento de pessoas. Os autores desta dissertação não têm conhecimento de que o P-CMM já esteja sendo utilizado no Brasil.

NÍVEIS DE MATURIDADE	TRILHAS DE ÁREA DE PROCESSO			
	DESENVOLVENDO CAPACIDADES INDIVIDUAIS	**CONSTRUINDO EQUIPES DE TRABALHO E CULTURA**	**MOTIVANDO E GERENCIANDO O DESEMPENHO**	**DESENVOLVENDO RECURSOS**
NÍVEL 5 – OTIMIZADO	Desenvolvimento contínuo das capacidades		Alinhamento do Desempenho Organizacional	Inovação Contínua dos Recursos
NÍVEL 4 – PREVISÍVEL	Ativos baseados em competências *Mentoring*[14]	Integração de competências Equipes de trabalho motivadas	Gerenciamento Quantitativo de Desempenho	Gerenciamento da Capacidade Organizacional
NÍVEL 3 – DEFINIDO	Desenvolvimento de Competências Análise de Competências	Desenvolvimento de equipes Cultura participativa	Práticas Baseadas em Competências Desenvolvimento da Carreira	Planejamento de Recursos
NÍVEL 2 – REPETÍVEL	Treinamento e Desenvolvimento	Comunicação e Coordenação	Recompensa Gerenciamento de Desempenho Ambiente de Trabalho	Recrutamento e Seleção[15]

Tabela 3. Estrutura do P-CMM[CURTIS et alli, 2001].

Resumidamente, os níveis evolutivos do P-CMM podem ser assim descritos:

- **Nível 1 – Inicial:** As organizações não conhecem sua real capacidade de trabalho e não conseguem identificar e desenvolver o potencial de seu pessoal de forma sistemática. Pouca integração entre os objetivos individuais e os objetivos do negócio, comunicação fraca e objetivos de desempenho imprecisos são alguns dos problemas. Como a troca de pessoal é freqüentemente alta, é difícil desenvolver habilidades.
- **Nível 2 – Repetível:** A empresa assume que o desenvolvimento contínuo de seus empregados é um valor importante e documenta estas novas políticas. Há processos bem definidos para o recrutamento, seleção e contratação de

[14] Optou-se por manter o termo original em inglês, uma vez que vem sendo utilizado no mercado nacional.
[15] Do original, em inglês: *staffing*.

talentos. Há mecanismos para garantir uma comunicação bidirecional e os objetivos de desempenho passam a ser claramente definidos.

- **Nível 3 – Definido:** Práticas adotadas localmente são estendidas para a organização. O planejamento do desenvolvimento da equipe é feito com base na análise dos processos de negócio e das competências principais envolvidas. São desenvolvidos: inventário das habilidades, plano individual de carreira, remuneração baseada no desenvolvimento, comunicação efetiva, plano de participação na tomada de decisões.
- **Nível 4 – Previsível[16]:** Times de alto desempenho compostos por pessoas com conhecimento e habilidades complementares, passando a ter vantagens competitivas no mercado. Atividades de formação de equipes[17] são as principais preocupações. O desempenho é previsível porque as competências essenciais são conhecidas quantitativamente. Um programa de orientação existe para auxiliar na resolução de conflitos, na tomada de decisão sobre a carreira, na melhoria do desempenho e desenvolvimento de novas habilidades.
- **Nível 5 – Otimizado:** A melhoria contínua está completamente instalada na empresa. Treinadores[18] provêm assistência especializada para melhorar o desempenho. Inovações individuais mais promissoras são disseminadas por toda a organização.

2.1.6 Outros modelos derivados

A partir da consagração dos modelos desenvolvidos pelo SEI, outros modelos começaram a surgir, utilizando os mesmos princípios de maturidade, porém focando segmentos específicos de atuação. Este é o caso, por exemplo, do *Trillium*, que é um modelo desenvolvido pela Bell Canadá [CANADA, 1992] para avaliar principalmente o desenvolvimento de produtos do ramo das telecomunicações. Também é um modelo em níveis evolutivos de maturidade, baseado nos conceitos usados no modelo SW-CMM e na norma ISO/IEC 9000-3.

---

[16] Na versão 1.0 o termo "gerenciado" era utilizado. Na versão 2.0, ele foi substituído pelo termo "previsível".
[17] Do original, em inglês: *team building.*
[18] Do original, em inglês: *coaches.*

2.2 Modelos de Qualidade de Processo de Software em nível pessoal e equipes

Além dos modelos de maturidade aplicáveis em nível organizacional, o SEI vem atuando no desenvolvimento, divulgação e disseminação de modelos de processos dedicados ao indivíduo ou a pequenas equipes de projeto, como:

- *Personal Software Process* (PSP)
- *Team Software Process* (TSP)

Paralelamente ao SEI, e utilizando conceitos complementares, outro modelo vem ganhando destaque no mercado, resultante da associação dos precursores das metodologias orientadas a objetos Ivar Jacobson, Grady Booch e James Rumbaugh:

- *Rational Unified Process* (RUP)

2.2.1 Personal Software Process (PSP)

O *Personal Software Process* (PSP) foi desenvolvido por Watts Humphrey e apresentado detalhadamente em seu livro *A Discipline for Software Engineering* [HUMPHREY, 1995]. Posteriormente, uma versão simplificada foi apresentada no livro *Introduction to the Personal Software Process* [HUMPHREY, 1997] e, mais recentemente, o SEI publicou um relatório técnico [HUMPHREY, 2000a] oferecendo uma visão geral do modelo.

O que o PSP oferece é um caminho para melhorar a qualidade, a previsibilidade e a produtividade, endereçando as necessidades de pequenas empresas e pequenas equipes de projeto. É uma abordagem sistemática para o desenvolvimento de software, baseada em métricas, planejamento e técnicas de engenharia de software.

A estrutura do PSP foi baseada no SW-CMM do qual Humphrey selecionou um conjunto de 12, das 18 KPAs, adaptando-as para o nível individual, de modo a auxiliar os engenheiros de software a gerenciar seu trabalho, avaliar seus talentos e construir suas habilidades. Foram deixadas de lado aquelas KPAs que, ou não são aplicáveis no nível individual, ou seus resultados não podem ser demonstrados para indivíduos [HUMPHREY, 1995], quais sejam: Gerenciamento de Subcontratos de Software, Coordenação Intergrupos, Gerenciamento de Requisitos, Gerenciamento de Configuração de Software, Garantia da Qualidade de Software e Programa de Treinamento.

O principal objetivo do PSP é mostrar aos engenheiros de software que um processo definido e mensurável pode auxiliá-los a melhorar o seu desempenho pessoal. À medida que o desempenho individual melhora, a *performance* de suas equipes e projetos também fica mais sujeita a melhorar ([HUMPHREY, 1994], [HUMPHREY, 1995]).

O PSP é introduzido em um curso regular com duração de 100-150 horas ao longo das quais são executados 10 exercícios e produzidos 5 relatórios. Ao final do treinamento, os próprios participantes podem constatar o quão útil pode ser um processo de software definido e mensurável, pois comparam seus próprios dados à medida que avançam nos níveis do PSP.

Conforme relatado em [HAYES, 1998], as estimativas tendem a ser mais otimistas quando baseadas intuitivamente no passado, pois os indivíduos tendem a lembrar-se de sua melhor *performance*. À medida que os engenheiros de software armazenam dados reais sobre seus projetos, e baseiam suas estimativas nestes dados, eles se tornam mais precisos e realistas em suas previsões.

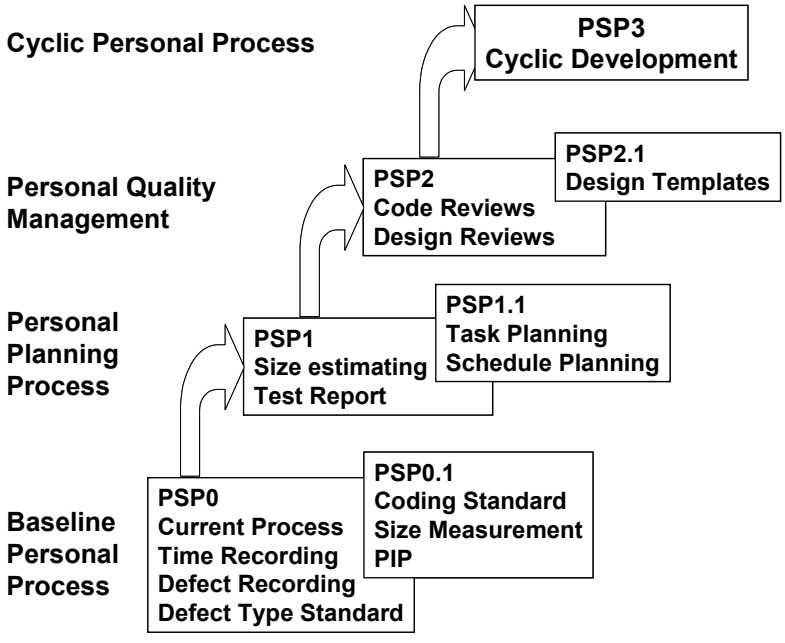

Figura 2. ***Personal Software Process*** **(PSP) [HUMPHREY, 1995]**.

Conforme mostrado na Figura 2, o PSP é estruturado em sete níveis de maturidade de processo através dos quais determinadas práticas de engenharia de

software são introduzidas. À medida que os indivíduos progridem na utilização do método, mais dados vão sendo armazenados, permitindo que eles mesmos acompanhem sua *performance* e o progresso de suas atividades. Isto lhes permite descobrir quais são as técnicas que mais se aplicam a si próprios em função de suas habilidades pessoais e tarefas. Como as práticas são introduzidas passo a passo, isto minimiza o impacto da mudança do processo no engenheiro de software.

Por ser o objeto deste trabalho, o método PSP será completamente detalhado no capítulo 4 desta dissertação.

2.2.2 Team Software Process (TSP)

Ao longo de seu trabalho com o PSP, Humphrey foi percebendo que durante os cursos de PSP os engenheiros de software compreendiam e utilizavam as práticas ensinadas. Porém, à medida que retomavam seu trabalho cotidiano e que se deparavam com um ambiente não propício à disciplina do PSP, acabavam abandonando o seu uso. Desta forma, surgiu a necessidade de desenvolver processos que envolvessem, não apenas o indivíduo, como também sua equipe e seus superiores.

Com este objetivo em mente, em 1996, Watts S. Humphrey, sob o patrocínio do SEI, iniciou o desenvolvimento do TSP - *Team Software Process.* Aos poucos, o TSP foi sendo testado e ajustado, resultando na versão atualmente disponível, cujo objetivo principal é guiar o desenvolvimento e a manutenção de projetos de software através de times de alta *performance*, utilizando recursos treinados em PSP [HUMPHREY, 1998].

Uma versão simplificada do modelo, TSPi, pode ser obtida em *Introduction to the Team Software Process* [HUMPHREY, 1999]. No entanto, por ter sido escrito principalmente para servir como um livro básico para ser usado em universidades, seu texto faz diversas referências ao papel do instrutor. É ele quem resolve toda a sorte de problemas: desde decidir sobre requisitos do projeto até solucionar questões relativas ao desempenho de determinado membro da equipe. Informações adicionais são necessárias para a aplicação do TSPi em larga escala industrial, conforme citado pelo próprio autor. Uma visão geral dobre o TSP pode ser encontrada em [HUMPHREY, 2000b] e relatos de sua utilização na indústria, em [MC ANDREWS, 2000].

O modelo de ciclo de vida usado pelo TSPi é o cíclico, ou seja, o produto vai sendo desenvolvido em partes, através de ciclos completos, conforme ilustra a Figura 3.

Ao final de cada ciclo, um produto testável é produzido. Cada ciclo é composto por sete fases: estratégia, planejamento, requisitos, projeto, implementação, teste e autópsia, além de uma etapa inicial denominada lançamento. A saída de um ciclo representa a entrada para o ciclo seguinte.

Os formulários, em um total de 21, auxiliam o registro de informações e o desenvolvimento das atividades. Alguns deles, como o formulário de registro de tempos (LOGT) e o formulário de registro de defeitos (LOGD), entre outros, são provenientes do PSP.

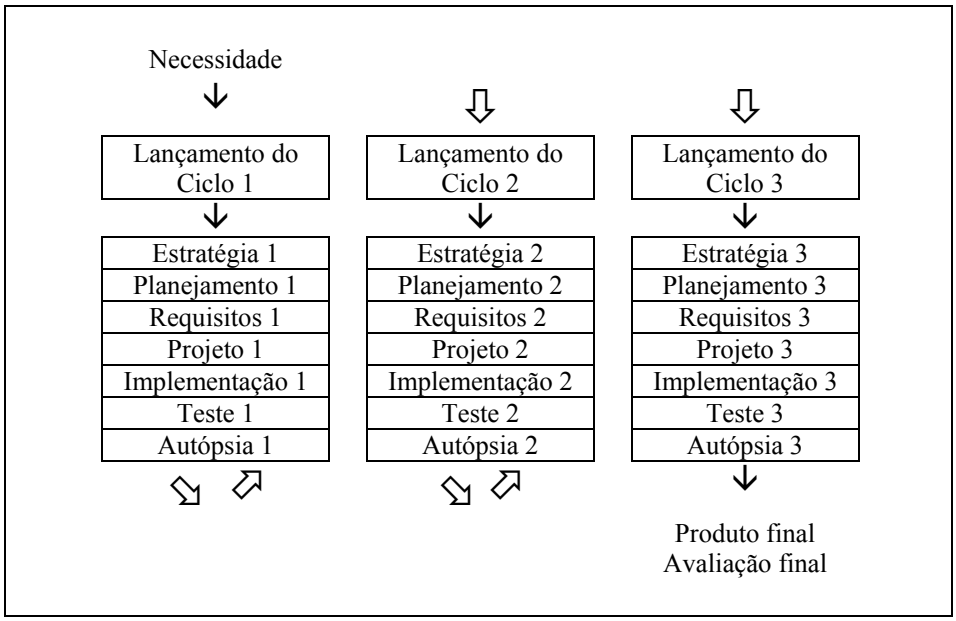

Figura 3. ***Team Software Process*** **(TSP) [HUMPHREY, 1999].**

Um ponto bastante positivo do TSPi é o fato de definir claramente os cinco papéis a serem desempenhados pelos membros da equipe, de modo a garantir seu bom funcionamento: Líder de Projeto, Gerente de Planejamento, Gerente de Desenvolvimento, Gerente de Processo/Qualidade e Gerente de Suporte. Cada um destes papéis é descrito, não somente em termos de perfil e características, como também em metas/métricas e principais atividades. Ao final do projeto, não apenas os membros do time terão sido avaliados segundo os critérios estabelecidos, como também os papéis por eles desempenhados.

Uma das principais críticas ao modelo PSP apresentada por Anne Disney em sua dissertação de mestrado [DISNEY & JOHNSON, 1998], e reforçado por todos os

relatos de experiência com o PSP[19], é a quantidade de informações que o engenheiro de software tem que registrar e a ausência de uma ferramenta automatizada para realizar as atividades de cálculo e análise, gerando distorções na interpretação dos resultados. Isto foi resolvido no TSPi com a disponibilização de uma ferramenta protótipo desenvolvida em Excel, por James W. Over, do SEI. Esta versão pode ser obtida no site da editora Adisson-Wesley.

Em [MC ANDREWS, 2000], o autor relata a utilização experimental do TSP nas empresas: Advanced Information Services, Teradyne, Boeing e Hill Air Force Base. Esta última, a primeira empresa do governo americano a ser avaliada como nível 5 do SW-CMM.

2.2.3 Rational Unified Process (RUP)

O RUP – *Rational Unified* Process é um processo de engenharia de software que provê uma forma disciplinada de atribuir responsabilidades e atividades em uma organização que desenvolve software [KRUCHTEN, 2000].

Originalmente concebido como uma unificação das abordagens orientadas a objetos de Ivar Jacobson, Grady Booch e James Rumbaugh, o RUP é hoje um conjunto coerente das melhores práticas de desenvolvimento de software que incluem:

- Desenvolvimento de software de forma iterativa e incremental: o produto vai sendo construído em etapas ou ciclos, que produzem um incremento no produto de software.
- Gerência de requisitos: envolve tanto a captura, elicitação, organização e comunicação, quanto o gerenciamento propriamente dito dos requisitos funcionais e não funcionais do produto de software.
- Arquitetura e uso de componentes: as atividades de projeto giram em torno da arquitetura e do uso extensivo de componentes.
- Modelagem visual: o RUP utiliza a UML (*Unified Modeling Language*) como padrão para representação de seus modelos.
- Verificação contínua da qualidade: a qualidade no RUP é verificada continuamente, tanto para o produto, quanto para o processo.

[19] Essas abordagens são amplamente discutidas no capítulo 3 desta dissertação.

- Gerenciamento de configuração e mudanças: especialmente devido ao fato que o desenvolvimento é iterativo e incremental, gerenciar adequadamente a configuração e as mudanças passa a ser um aspecto crucial tratado pelo RUP.

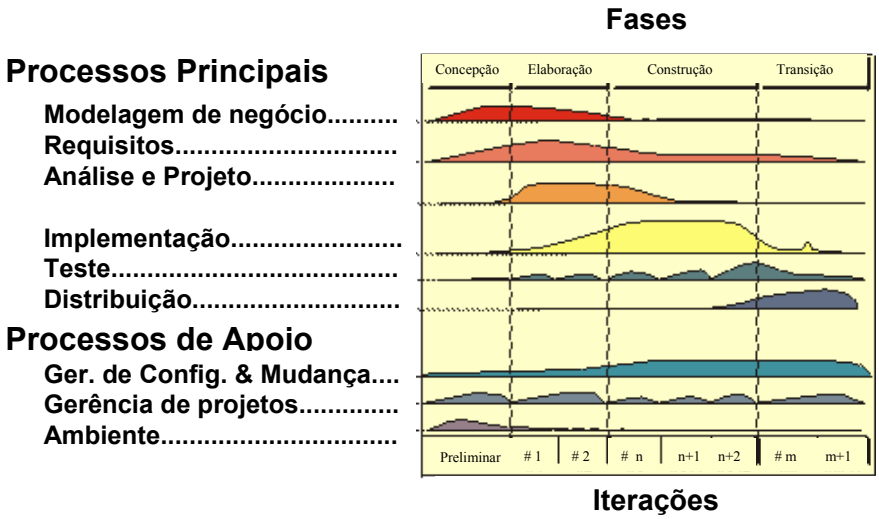

Figura 4. *Rational Unified Process* (RUP) [KRUCHTEN, 2000].

Conforme se pode observar na Figura 4, o RUP é composto por duas dimensões. A dimensão horizontal, representando a evolução ao longo do tempo, compreende as fases do ciclo: concepção, elaboração, construção e transição. A dimensão vertical, que agrupa as atividades de acordo com sua natureza: modelagem de negócios, requisitos, análise e projeto, implementação, testes, distribuição; além das disciplinas de apoio: gerência de configuração e mudanças, gerência de projetos e ambiente.

O RUP já apresenta evoluções em relação ao seu antecessor o UP – *Unified Process*, do qual é derivado. Porém, para tornar-se um processo completo, que englobe todos os aspectos que envolvem a produção de software, ainda há pendências a serem solucionadas.

Em [MACHADO et alli, 2000], os autores exploram os aspectos da norma NBR ISO/IEC 12207[20] que ainda não são cobertos pelo RUP, ou o são de forma parcial, como os processos de: aquisição, fornecimento, operação, garantia da qualidade, verificação, validação, auditoria, resolução de problemas, infra-estrutura, melhoria, treinamento e adaptação.

2.3 Esforços de padronização na área de software

A *Internacional Organization for Standardization* (ISO) é uma organização não governamental estabelecida em 1947, cuja missão é promover o desenvolvimento da normalização e atividades relacionadas, em nível mundial. O seu trabalho consiste em estabelecer padrões, que se tornam Normas Internacionais, através de consenso entre membros participantes de Comitês Técnicos nos mais diversos países. A *International Electrotechnical Commission* (IEC) é uma organização mundial, fundada em 1906, que publica Normas Internacionais relacionadas à eletricidade e à eletrônica.

De modo a atender a interesses comuns, essas duas organizações, ISO e IEC, estabeleceram um comitê conjunto denominado *Joint Technical Committee 1* (JTC1) para elaboração de normas na área de Tecnologia da Informação. Neste Comitê as normas relativas a *software* são trabalhadas por grupos de trabalho, *Working Groups* (WG), subordinados ao *Software Committe 7- Software Engineering* (SC7). Cada um desses WGs atua sobre um segmento específico relacionado à engenharia de software.

No Brasil, o órgão responsável pela normalização é a *Associação Brasileira de Normas Técnicas* (ABNT). Na ABNT também existe um Comitê específico para software (CB21) sob o qual estão estruturadas Comissões de Estudo (CE), correspondentes à cada um dos WGs do JTC1/SC7.

As normas que se relacionam à qualidade do processo de *software* que são de interesse da engenharia de software são:

- NBR ISO/IEC 12207:1998 – Tecnologia da Informação – Processos de Ciclo de Vida de Software [ABNT, 1998] (correspondente à norma internacional ISO/IEC 12207:1995).
- NBR ISO 9000-3:1997 – Normas de Gestão da Qualidade e Garantia da Qualidade – parte 3: Diretrizes para aplicação da NBR ISO 9001 ao

[20] A NBR ISO/IEC 12207 é discutida com mais detalhes na próxima seção.

desenvolvimento, fornecimento e manutenção de software [ABNT, 1997] (correspondente à norma internacional ISO 9000-3:1997).

- ISO/IEC TR 15504[21] – *Information Technology - Software Process Assessment* [ISO/IEC, 1998b].

Há ainda as normas dedicadas à definição e avaliação da qualidade do produto de software, que, no entanto, não serão tratadas por este trabalho, conforme explicado anteriormente. São elas:

- NBR 13596:1996 – Tecnologia de informação - Avaliação de produto de software – Características de qualidade e diretrizes para seu uso (correspondente à norma internacional ISO/IEC 9126:1991). Para maiores detalhes consultar [ISO/IEC, 1991] e/ou [ABNT, 1996].
- ISO/IEC 14598-5[22] – *Information Technology - Software product evaluation (6 parts)*. Para maiores detalhes consultar [ISO/IEC, 1998a] e [ISO/IEC, 1999].

2.3.1 NBR ISO/IEC 12207:1998

A norma internacional ISO/IEC 12207 foi publicada em 1995 e sua correspondente nacional NBR ISO/IEC 12207 [ABNT, 1998], em 1998. Ela utiliza uma terminologia bem definida para estabelecer os processos, atividades e tarefas que devem ser aplicados durante a aquisição, fornecimento, desenvolvimento, operação e manutenção de software. Cada processo é definido em termos de suas atividades, e cada atividade, por sua vez, é definida em termos de suas tarefas. Os processos de ciclo de vida estão agrupados em três classes, conforme ilustra a Tabela 4:

- **Fundamentais:** atendem o início e a execução do desenvolvimento, operação ou manutenção de produtos de software.
- **Apoio:** auxiliam um outro processo e contribuem para o sucesso e a qualidade do projeto de software.
- **Organizacionais:** são empregados pela organização para estabelecer e implementar uma estrutura subjacente para melhorar continuamente a estrutura e os processos. Eles são tipicamente empregados fora do domínio de projetos e contratos específicos.

[21] A ISO/IEC TR 15504 ainda não tem sua equivalente nacional.

CLASSE	PROCESSO
FUNDAMENTAIS	Aquisição Fornecimento Desenvolvimento Operação Manutenção
APOIO	Documentação Gerência de Configuração Garantia da Qualidade Verificação Validação Revisão Conjunta Auditoria Resolução de Problema
ORGANIZACIONAIS	Gerência Infra-estrutura Melhoria Treinamento
ADAPTAÇÃO	

Tabela 4. Processos da Norma NBR ISO/IEC 12207 [ABNT, 1998].

Dois princípios básicos foram adotados na concepção da arquitetura do ciclo de vida de software: modularidade e responsabilidade. Modularidade, no sentido de que os processos são fortemente coesos e fracamente acoplados. Responsabilidade, no sentido de que cada processo tem um responsável, que é uma das partes envolvidas.

A Norma é bastante flexível no que diz respeito aos princípios de engenharia de software, podendo ser utilizada com qualquer modelo de ciclo de vida (cascata, incremental, evolutivo, prototipação, RAD etc.) e com qualquer linguagem de programação. Estas decisões são deixadas a cargo de quem a utilizará. Além disto, possui um Guia [ISO/IEC, 1998c] para facilitar a sua aplicação e adaptação. Este Guia encontra-se em fase de publicação pela ABNT.

A Norma internacional ISO/IEC 12207:1995 encontra-se em fase de revisão, de modo a tornar-se mais aderente aos requisitos de avaliação que estão sendo estabelecidos no relatório técnico ISO/IEC TR 15504. Os itens revisados serão publicados na forma de uma extensão à norma original e encontra-se em fase de publicação pela ISO.

---

[22] A ISO/IEC 14598 ainda não tem sua equivalente nacional.

2.3.2 NBR ISO 9000-3:1997

A norma internacional ISO 9000-3, e a sua correspondente nacional NBR ISO 9000-3:1997 [ABNT, 1997], ao contrário da ISO 9001, não são normas certificadoras e, sim, guias que oferecem diretrizes para aplicação da ISO 9001 ao desenvolvimento, fornecimento e manutenção de software.

As diretrizes propostas cobrem questões como: entendimento comum para as partes sobre requisitos funcionais e o uso de metodologias consistentes para desenvolvimento de software e gerenciamento do projeto como um todo (da concepção até a manutenção do software). A ISO 9000-3 é dividida em 3 partes principais:

- *Estrutura do Sistema da qualidade*: descreve aspectos organizacionais, relacionados ao sistema de qualidade: responsabilidade do fornecedor, responsabilidade do comprador, análise crítica conjunta.
- *Atividades do ciclo de vida*: descreve as atividades de desenvolvimento de software: análise crítica do contrato, especificação dos requisitos do comprador, planejamento do desenvolvimento, projeto e implementação, testes e validação, aceitação, cópia, entrega e instalação, manutenção.
- *Atividades de apoio* (não dependentes da fase): descreve as atividades que apóiam o ciclo de vida de desenvolvimento: gerenciamento da configuração, controle de documentos, registros de qualidade, medição, regras e convenções, ferramentas e técnicas, aquisição, produto de software incluído e treinamento.

Atualmente, a ISO 9000-3 está sendo revisada de modo a acomodar as modificações que a própria norma certificadora ISO 9001 sofreu com a liberação da versão 2000. Pela primeira vez essa tarefa está a cargo do JTC1/SC7. Ou seja, ao contrário da versão anterior, são os profissionais de software que estarão revisando a ISO 9000-3 com o objetivo de apropriar e relacionar os conhecimentos consolidados nas demais normas ISO referentes a software como a ISO/IEC 12207, ISO/IEC TR 15504, ISO/IEC 9126 e outras.

2.3.3 ISO/IEC TR 15504

Em janeiro de 1993, a ISO, através do ISO/IEC JTC1, motivada pela necessidade emergente de uma padronização internacional para avaliação de processos

de software, constituiu o projeto SPICE - *Software Process Improvement and Capability dEtermination*, cujos objetivos iniciais eram:

- Desenvolver uma versão preliminar para avaliação de processos de software, mais geral e abrangente que os modelos existentes;
- Promover experiências com usuários para coletar dados para servir de base para a revisão antes de sua publicação como norma internacional;
- Conscientizar o mercado da existência do novo padrão de avaliação de processo de software e promover o seu uso.

O resultado deste esforço conjunto de diversos países foi a publicação do relatório técnico ISO/IEC TR 15504 [ISO/IEC, 1998b], constituído nesta versão por nove documentos. O objetivo do grupo é publicá-lo como norma internacional ISO/IEC em 2002.

O relatório técnico ISO/IEC TR 15504 é um *framework* para avaliação de processos de software que cumpre dois objetivos: a melhoria dos processos e a determinação da capacidade de processos de uma organização. Para isto, ele está estruturado em duas dimensões: os processos e o seu nível de capacidade.

Conforme visto nas seções 2.1.1 e 2.1.3, no SW-CMM se avaliava o nível de maturidade de uma organização baseado no fato dela ter ou não implementadas determinadas habilidades (KPAs). Ou seja, uma organização estava no nível 2 do SW-CMM se tivesse implementadas todas as KPAs referentes àquele nível. Já no CMMI, a exemplo, e por influência, da ISO/IEC 15504, os processos podem ser avaliados separadamente, conforme o plano de melhorias da organização.

Os níveis segundo os quais são avaliados os processos na ISO/IEC TR 15504 são os seguintes [ROCHA, MALDONADO & WEBER, 2001]:

- **Nível 0 – Incompleto:** ou o processo não está implementado, ou não oferece evidências de que os produtos sejam consistentemente produzidos.
- **Nível 1 – Executado:** o processo alcança seus propósitos, porém não existem evidências de que seja adequadamente acompanhado e gerenciado.
- **Nível 2 – Gerenciado:** o processo gera os produtos de acordo com o que foi planejado em termos de tempo e recursos.

- **Nível 3 – Estabelecido:** o processo é uma instância de um processo padrão que foi definido segundo princípios de engenharia de software.
- **Nível 4 – Previsível:** o processo é quantitativamente previsível e é executado dentro de limites definidos para obter seus resultados.
- **Nível 5 – Em otimização:** o processo encontra-se em constante evolução sendo melhorado para atingir os objetivos de negócio atuais e futuros da organização.

2.4 Conclusões sobre os modelos

Conforme abordado neste capítulo, diversos têm sido os esforços em nível mundial para a padronização da definição e da avaliação dos processos que envolvem a questão de software. Alguns destes modelos são aplicáveis em nível organizacional, como a família de normas ISO/IEC e os modelos da família CMM. Outros são aplicáveis em nível pessoal e de pequenas equipes, como o PSP, TSP e o RUP. Alguns possuem o seu foco nos processos de produção e aquisição, como o SW-CMM, SA-CMM, CMMI, ISO/IEC 12207 e a ISO/IEC TR 15504. Outros, possuem o seu foco nas práticas pessoais, como o P-CMM, RUP, TSP e PSP. Porém, o que se evidencia em todos eles, em menor ou maior grau, é a preocupação constante em integrar os 5 “P”s do desenvolvimento de software apresentados no capítulo anterior.

2.5 A opção pelo PSP

Ao selecionar o PSP como sendo o modelo a ser tomado como base para o trabalho desta dissertação, levou-se em consideração suas principais características, detalhadas a seguir:

- O PSP é um processo completo para o desenvolvimento de software que introduz gradativamente: práticas de engenharia de software, métricas e análise das métricas;
- O PSP é um processo escalonável em relação aos níveis de maturidade, podendo ser usado por desenvolvedores menos e mais experientes, por equipes menos e mais maduras e por organizações menos ou mais organizadas;

- O PSP é um processo escalonável em relação à abrangência, podendo iniciar com um indivíduo ou uma pequena equipe de projeto e ser expandido para ser usado por toda a organização;
- O PSP é um processo genérico em relação ao escopo e ao domínio de aplicações, podendo ser utilizado em projetos de manutenção de ambientes legados ou novos desenvolvimentos, desde que providas as respectivas adaptações;
- O PSP implanta a base necessária para a adoção de outros modelos de melhoria em nível organizacional como os modelos da família CMM ou as normas ISO;
- O PSP introduz uma disciplina de desenvolvimento ao codificador que possibilita que ele esteja alerta para as questões de qualidade desde o início do desenvolvimento;
- O PSP apesar de ter um certo grau de generalidade, já é mais específico que os demais modelos, prescrevendo, não apenas o que deve ser feito, mas também como deve ser feito.

Essas características do PSP poderão ser melhor compreendidas através do detalhamento do modelo que é apresentado no próximo capítulo.

2.6 Considerações sobre o capítulo

Neste capítulo foram apresentados e analisados os modelos de definição e melhoria de processos, bem como as normas internacionais que estão sendo desenvolvidas com a mesma finalidade. Foi ainda introduzido, de forma superficial, o modelo PSP, que é a base desta dissertação e que será detalhado no próximo capítulo. Ao final, foram também apresentados os motivos que levaram à escolha do PSP.

CAPÍTULO 3 - O MODELO PSP

Conforme introduzido no capítulo 2, e ilustrado na Figura 2 (pág. 25), o PSP – *Personal Software Process* – está estruturado em sete níveis evolutivos de maturidade, organizados em quatro grupos de processos:

- *Baseline Personal Process* (PSP0 e PSP0.1): processos básicos que visam introduzir as métricas iniciais: tempo, defeitos e tamanho.
- *Personal Planning Process* (PSP1 e PSP1.1): processos que introduzem as ferramentas de planejamento, como o método de estimativas PROBE e as orientações para planejar e acompanhar o cronograma.
- *Personal Quality Management* (PSP2 e PSP2.1): grupo de processos que introduz os conceitos de qualidade através de revisões e ferramentas para projeto conceitual do programa.
- *Cyclic Personal Process* (PSP3): processo que pretende estender os conceitos do PSP para projetos maiores.

Naquele mesmo capítulo também foi visto que, para auxiliar a introdução, o aprendizado e a utilização do PSP, o modelo foi organizado em forma de roteiros, formulários e padrões.

Os roteiros do PSP, apresentados na Tabela 5, funcionam como um guia para a execução dos processos, oferecendo orientações detalhadas passo a passo para cada uma das etapas que o engenheiro de software deve seguir. Além disto, os roteiros informam quais são os critérios de entrada, os critérios de saída e os artefatos produzidos em cada uma destas etapas.

Conforme se pode observar na Tabela 5, cada nível possui um roteiro geral, denominado *Process Script*, e roteiros específicos para cada uma das fases do projeto: *Planning Script*, *Development Script* e *Postmortem Script*. O roteiro geral, como o próprio nome diz, oferece uma visão geral dos três outros roteiros. Processos mais maduros incorporam técnicas mais refinadas para cada uma das fases, sendo refletidas na sofisticação, também maior, de seus roteiros. Processos menos maduros, por sua vez,

possuem roteiros mais básicos e com menor quantidade de informações a serem preenchidas.

GRUPO	NÍVEL	CÓDIGO	ROTEIRO
BASELINE PROCESS	PSP0	C10	*PSP0 Process Script*
		C11	*PSP0 Planning Script*
		C12	*PSP0 Development Script*
		C13	*PSP0 Postmortem Script*
	PSP0.1	C21	*PSP0.1 Process Script*
		C22	*PSP0.1 Planning Script*
		C23	*PSP0.1 Development Script*
		C24	*PSP0.1 Postmortem Script*
PERSONAL PLANNING PROCESS	PSP1	C30	*PSP1 Process Script*
		C31	*PSP1 Planning Script*
		C32	*PSP1 Development Script*
		C33	*PSP1 Postmortem Script*
		C36	*PROBE Estimating Script*
	PSP1.1	C41	*PSP1.1 Process Script*
		C42	*PSP1.1 Planning Script*
		C43	*PSP1.1 Development Script*
		C44	*PSP1.1 Postmortem Script*
PERSONAL QUALITY MANAGEMENT	PSP2	C51	*PSP2 Process Script*
		C52	*PSP2 Planning Script*
		C53	*PSP2 Development Script*
		C54	*PSP2 Postmortem Script*
	PSP2.1	C59	*PSP2.1 Process Script*
		C60	*PSP2.1 Planning Script*
		C61	*PSP2.1 Development Script*
		C62	*PSP2.1 Postmortem Script*
CYCLIC PERSONAL PROCESS	PSP3	C74	*PSP3 Process Script*
		C75	*PSP3 Planning Script*
		C76	*PSP3 High-Level Design Template*
		C77	*PSP3 High-Level Design Review Script*
		C78	*PSP3 Development Script*
		C79	*PSP3 Postmortem Script*

Tabela 5. Roteiros do PSP.

Os formulários e padrões, descritos na Tabela 6, bem como os respectivos roteiros de preenchimento, oferecem uma forma organizada e estruturada de coletar e analisar as métricas de processo e de produto coletadas ao longo do desenvolvimento do programa. Conforme se pode observar na Tabela 6, a cada nível superior de maturidade, o formulário destinado a armazenar os dados do projeto, *Project Plan Summary*, vai sendo refinado e completado, dando origem a novas versões (*PSP0 – Project Plan Summary, PSP0.1 – Project Plan Summary, PSP1 – Project Plan Summary* etc.). Já os

demais formulários, vão sendo incorporados ao processo do engenheiro de software e não sofrem mais alterações a cada novo nível.

GRUPO	NÍVEL	CÓDIGO	FORMULÁRIO/PADRÃO
BASELINE PROCESS	PSP0	C14	*PSP0 Project Plan Summary*
		C16	*Time Recording Log*
		C18	*Defect Recording Log*
		C20	*Defect Type Standard*
	PSP0.1	C25	*PSP0.1 Project Plan Summary*
		C27	*PIP – Process Improvement Proposal*
		C29	*C++ Coding Standard*
PERSONAL PLANNING MANAGEMENT	PSP1	C34	*PSP1 Project Plan Summary*
		C37	*Test Report Template*
		C39	*Size Estimating Template*
	PSP1.1	C45	*PSP1.1 Project Plan Summary*
		C47	*Task Planning Template*
		C49	*Schedule Planning Template*
PERSONAL QUALITY MANAGEMENT	PSP2	C55	*PSP2 Project Plan Summary*
		C57	*C++ PSP2 Design Review Checklist*
		C58	*C++ Code Review Checklist*
	PSP2.1	C63	*PSP2.1 Project Plan Summary*
		C65	*PSP2.1 Design Review Checklist*
		C66	*Operational Scenario Template*
		C68	*Functional Specification Template*
		C70	*State Specification Template*
		C72	*Logic Specification Template*
CYCLIC PERSONAL PROCESS	PSP3	C80	*PSP3 Project Plan Summary*
		C82	*Cycle Summary*
		C84	*PSP3 Design Review Checklist*
		C85	*Issue Tracking Log*

Tabela 6. Formulários e Padrões do PSP.

Neste capítulo será apresentada com mais detalhe a estrutura do PSP, bem como cada um de seus níveis, fazendo referência aos roteiros e formulários, quando necessário. Para uma descrição mais detalhada dos campos que compõem os formulários, bem como dos roteiros que guiam o seu preenchimento, consultar o apêndice C de [HUMPHREY, 1995]. Para uma introdução ao PSP em português, consultar [REINEHR, 2001].

3.1 PSP0 e PSP0.1– Baseline Personal Process

Os dois primeiros níveis do PSP, pertencentes ao grupo *Baseline Personal Process*, constituem a base para o processo de melhoria, introduzindo medições e relatórios padronizados. Neste momento, os engenheiros de software são encorajados a

empregar seus próprios métodos, porém seguindo uma seqüência pré-definida de fases, conforme ilustrado na Figura 5: planejamento, projeto, codificação, compilação, teste e autópsia[23]. Ao coletar dados sobre suas tarefas, começam a construir um entendimento quantitativo de cada passo do processo.

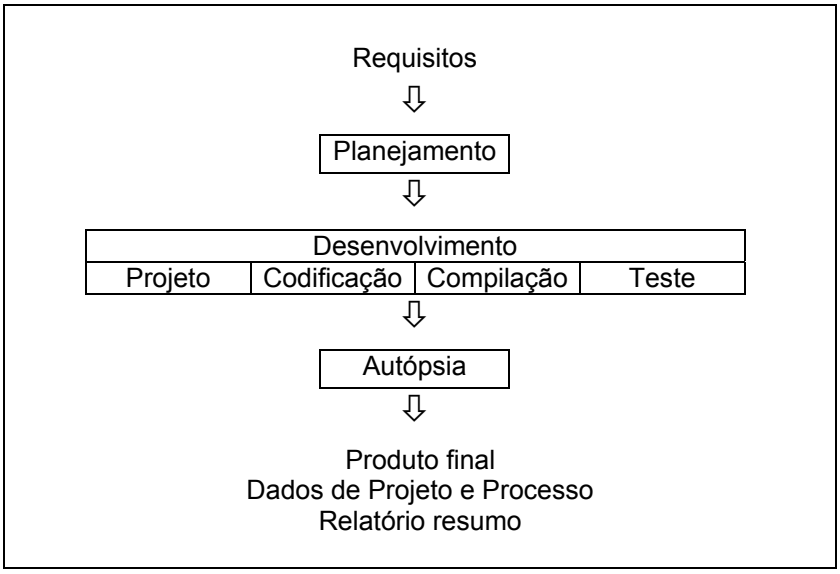

Figura 5. Processo original do PSP0 [HUMPHREY, 1995].

O modelo PSP trabalha com três métricas básicas: tempo, defeitos e tamanho. No PSP0 duas delas são introduzidas: tempo e defeitos. A terceira, tamanho, é apresentada no PSP0.1. As demais métricas que compõem o modelo são derivadas da combinação das métricas básicas.

A fase de planejamento do PSP0 é muito simples, resumindo-se à estimativa de tempo (em minutos) que o engenheiro prevê para completar o desenvolvimento do produto. Esta informação é registrada em um formulário de planejamento denominado *C14 – PSP0 Project Planning Summary.*

Baseado na afirmativa de Humphrey de que "você irá gastar seu tempo esta semana exatamente da forma como gastou na semana passada" [HUMPHREY, 1997], o PSP0 inicia mantendo um acompanhamento sobre o tempo dos engenheiros. Eles registram suas atividades, o tempo gasto com cada uma delas, bem como qualquer

[23] Do original, em Inglês: *postmortem.*

interrupção ou detalhe relevante, em um formulário denominado *C16 - Time Recording Log*.

Através do formulário *C18 – Defect Recording Log*, os desenvolvedores registram dados sobre os defeitos encontrados e o tempo correspondente para consertá-los. Uma padronização para os tipos de defeito é apresentada em [HUMPHREY, 1995].

Na fase de autópsia, ou seja, logo após a conclusão do desenvolvimento, os dados coletados são sumarizados no formulário de planejamento. No PSP0, este formulário é bastante simples, constituindo-se das medições totalizadas de tempo e defeito por fase. Porém, à medida que o processo evolui ao longo dos demais níveis do PSP, este formulário será alterado para contemplar novas informações, conforme demonstrado na Tabela 6, apresentada no início deste capítulo. Os formulários de tempo e defeitos, respectivamente C16 e C18, não serão alterados ao longo do processo.

Para atingir o PSP0.1, são acrescentados ao PSP0: um padrão de codificação, uma proposta de melhorias de processo (PIP) e a medição em LOC.

Para que a contagem das linhas de código seja consistente, Humphrey apresenta um padrão de codificação para C++ em [HUMPHREY, 1995]. Como este padrão é dependente da linguagem que está sendo utilizada, Albuquerque propõe um padrão para JAVA em [ALBUQUERQUE & MEIRA, 1997] e Silva propõe um padrão para Powerscript (a linguagem de programação do ambiente de desenvolvimento Powerbuilder) em [SILVA & MASIERO, 1998]. Além do padrão de codificação, é necessário ter um padrão de contagem de LOC propriamente dito. Isto também é sugerido por Humphrey em [HUMPHREY, 1995] para a linguagem C++ e por Silva em [SILVA & MASIERO, 1998] para o Powerscript.

No decorrer do desenvolvimento, o engenheiro de software pode ter idéias sobre como melhorar o processo. Estas idéias vão sendo registradas no formulário *C27 - Process Improvement Proposal (PIP),* que o acompanhará durante todos os níveis do PSP sem sofrer alterações.

Medidas mais detalhadas passam a fazer parte do processo, como: LOC inseridas, modificadas, deletadas e reusadas. Estas medições são registradas, na fase de autópsia, no formulário *C25 - PSP0.1 Project Plan Summary*, que é uma evolução do formulário de planejamento utilizado no PSP0 (C14).

3.2 PSP1 e PSP1.1– Personal Planning Process

Os dois próximos níveis do PSP têm os seus focos no planejamento pessoal do desenvolvimento, por isso estão agrupados sob o título de *Personal Planning Process*.

Para poder planejar de forma mais precisa e realista é necessário primeiramente conseguir estimar tamanho e esforço de forma mais efetiva. É na tarefa de estimar que se concentra o PSP1 e na tarefa de planejar, propriamente dita, que se concentra o PSP1.1.

No PSP1 Humphrey introduz o método de estimativa PROBE (*PROxy-Based Estimating*), que é o elemento central do PSP e que foi especialmente desenvolvido para ser por ele utilizado. O PROBE, ilustrado na Figura 6, foi baseado em princípios e técnicas conhecidos como a lógica *Fuzzy*, método do componente padrão, Análise por Pontos de Função (FPA) e técnicas estatísticas. O método utiliza o tamanho relativo de um *proxy*[24] para auxiliar na estimativa inicial, utilizando dados históricos para converter o tamanho do *proxy* em LOC estimadas.

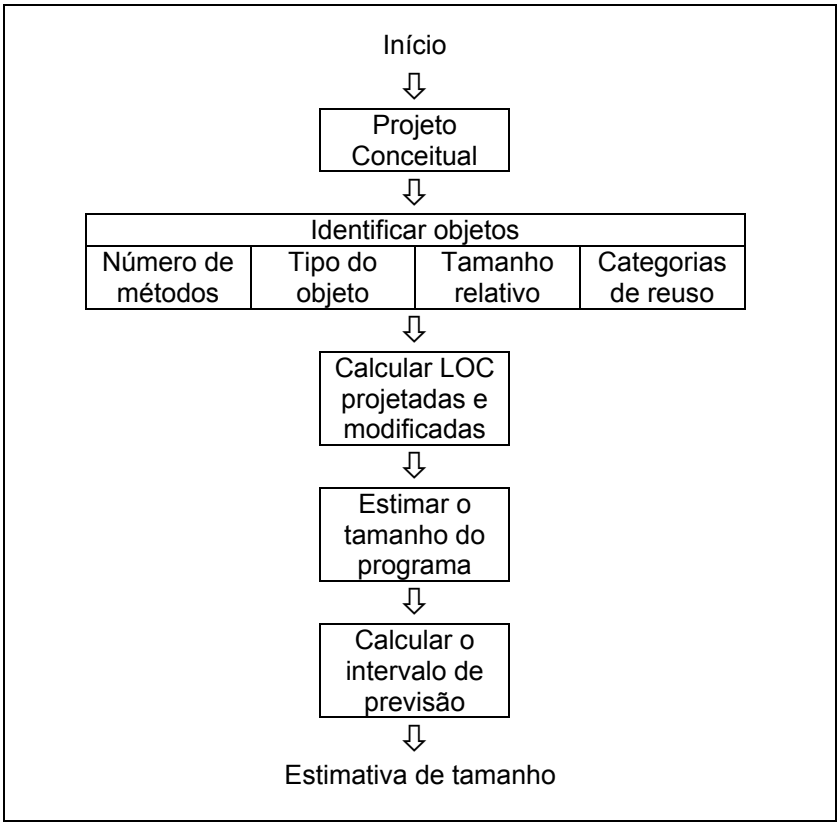

Figura 6. Método PROBE [HUMPHREY, 1995].

[24] Optou-se por manter a denominação original em inglês, uma vez que é bastante utilizada no Brasil.

O PROBE parte do projeto conceitual do programa, a partir do qual se deve identificar os prováveis objetos. A quantidade de LOC destes objetos é obtida utilizando uma tabela que relaciona: tamanho relativo do objeto (Ex.: muito pequeno, pequeno, médio, grande e muito grande) com o tipo do objeto (Ex.: cálculo, I/O, lógico etc.), fornecendo uma quantidade aproximada de LOC por método. Estes números devem, a rigor, ser obtidos a partir de uma base histórica do próprio desenvolvedor. Humphrey apresenta um referencial para C++, ilustrado na Tabela 7, e outro para Object Pascal, ilustrado na Tabela 8, para que o desenvolvedor utilize enquanto ainda não possui a sua própria base. A montagem das categorias de tamanho relativo é, de alguma forma, descrita em [HUMPHREY, 1995], porém, não faz parte diretamente do PSP e nem do método PROBE. O PROBE parte do princípio que essa tabela está pronta.

CATEGORIAS DE TAMANHO PARA C++ (LOC POR MÉTODO)					
	MUITO PEQUENO	**PEQUENO**	**MÉDIO**	**GRANDE**	**MUITO GRANDE**
CÁLCULO	2.34	5.13	11.25	24.66	54.04
DADOS	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
LÓGICA	7.55	10.98	15.98	23.25	33.83
SET-UP	3.88	5.04	6.56	8.53	11.09
TEXTO	3.75	8.00	17.07	36.41	77.66

Tabela 7. Categorias de tamanho relativo- C++ [HUMPHREY, 1995].

CATEGORIAS DE TAMANHO PARA OBJECT PASCAL (LOC POR MÉTODO)					
	MUITO PEQUENO	**PEQUENO**	**MÉDIO**	**GRANDE**	**MUITO GRANDE**
CONTROLE	4.24	8.68	17.79	36.46	74.71
DISPLAY	4.72	6.35	8.55	11.50	15.46
ARQUIVO	3.30	6.23	11.74	22.14	41.74
LÓGICA	6.41	12.42	24.06	46.60	90.27
IMPRESSÃO	3.38	5.86	10.15	17.59	30.49
TEXTO	4.63	8.73	16.48	31.09	58.62

Tabela 8. Categorias de tamanho relativo - Object Pascal [HUMPHREY, 1995].

Com o total de LOC estimado para os objetos, obtém-se o total estimado do programa aplicando-se princípios de estatística sobre os dados históricos. Se a quantidade de informações da base histórica for suficiente (três pontos ou mais

correlacionados), aplica-se a regressão linear. Em seguida, calcula-se os intervalos de previsão. A estimativa de tempo é calculada de forma similar. Detalhes da aplicação do método PROBE podem ser obtidos em [HUMPHREY, 1995] no roteiro *C36 – PROBE Estimating Script* e no formulário *C39 – Size Estimating Template.* Informações sobre o PROBE em português podem ser obtidas em [REINEHR, 2001].

Estas estimativas são registradas no formulário de planejamento: *C34 – PSP1 Project Plan Summary.* Este formulário será completado na fase de autópsia, após o desenvolvimento, com as informações totalizadas de tempo, defeitos e tamanho real do programa.

Ainda no PSP1, o formulário *C37 – Test Report Template* é introduzido para relatar a fase de teste com detalhes, permitindo ao engenheiro de *software*, não somente registrar informações sobre o planejamento e a execução dos testes, como também fazer testes de regressão mais seguros.

O nível PSP1.1 dedica-se efetivamente a auxiliar o desenvolvedor na tarefa de planejar o seu trabalho, uma vez que as estimativas de tamanho e esforço já foram incorporadas ao processo no nível anterior. Para planejar é necessário primeiramente distribuir o esforço estimado entre as diversas atividades do projeto e depois distribuir este esforço no tempo real disponível semanalmente para o projeto. Estas tarefas são facilitadas pelos formulários *C37 – Task Planning Template* e *C49 - Schedule Planning Template*, respectivamente. Ao utilizar a técnica do valor agregado[25], que atribui a cada atividade um valor agregado planejado baseado no percentual de esforço requerido, o método permite ao engenheiro de *software* gerenciar o progresso do projeto de forma mais real e efetiva.

O formulário de planejamento, neste nível, chama-se *C45 – PSP1.1 Project Plan Summary* e difere do anterior (C34) pela introdução do campo referente ao *CPI - Cost-Performance Index.* O *CPI* é dado pela razão entre o tempo total planejado acumulado e o tempo total real acumulado e expressa, portanto, o quanto o desenvolvedor está atingindo seus compromissos de custo. O ideal é que o *CPI* seja próximo a 1, refletindo que sua capacidade de planejamento está melhorando, uma vez que o tempo planejado está próximo ao tempo realmente gasto.

[25] Do original, em Inglês: *earned value.*

3.3 PSP2 e PSP2.1 – Personal Quality Management

O foco dos níveis PSP2 e PSP2.1 é o gerenciamento da qualidade pessoal. O PSP utiliza a técnica de revisão como o principal instrumento para melhorar a qualidade. O PSP2 introduz as Revisões de Código e de Projeto e o PSP2.1 introduz um padrão para projeto.

O custo da qualidade (COQ)[26] para o PSP é composto por dois elementos: o percentual de tempo gasto em Revisões de Código e de Projeto em relação ao tempo total do projeto e o percentual de tempo gasto em compilação e testes em relação ao tempo total do projeto. À medida que os desenvolvedores progridem na utilização do método, o tempo percentual gasto em revisões tende a aumentar, se comparado ao tempo gasto em compilação e testes, o que denota uma redução no custo de desenvolvimento.

O princípio básico adotado por Humphrey é o de que o passo mais importante dado para melhorar a qualidade de produto e o desempenho do engenheiro de software é a realização de revisões [HUMPHREY, 1995]. Para isto, ele apresenta um *checklist*[27], *C58 – C++ Code Review Checklist*, para a Revisão de Código (RC) de programas escritos em C++. Como esta etapa é extremamente dependente da linguagem de programação utilizada, Silva apresenta em [SILVA & MASIERO, 1998] um *checklist* para Powerbuilder. Albuquerque, no entanto, opta por aproveitar o *checklist* original de C++ para JAVA, sem modificações [ALBUQUERQUE & MEIRA, 1997].

Para auxiliar o desenvolvedor na tarefa de revisar o projeto, um roteiro básico, também na forma de *checklist*, é fornecido pelo PSP2: *C57 – C++ PSP2 Design Review Checklist*. Como o foco é quase exclusivamente na fase de codificação, a Revisão de Projeto (RP) foca a estrutura do programa e, portanto também é dependente de linguagem e acaba por quase se confundir com o próprio *chekclist* de projeto. Apesar disto, nem Albuquerque e nem Silva apresentam variantes para esse *checklist* em [ALBUQUERQUE & MEIRA, 1997] e [SILVA & MASIERO, 1998], respectivamente.

O formulário de planejamento do PSP2 passa a incorporar informações relativas às fases de Revisão de Código e Revisão de Projeto, bem como as taxas de eficiência da

[26] Do original, em Inglês: *Cost of Quality (COQ)*.
[27] Optou-se por manter a denominação original em inglês, *checklist*, uma vez que é bastante utilizada.

remoção de defeitos em cada fase, refletindo as alterações no processo ilustradas na Figura 7. Neste nível este formulário é chamado de *C55 – PSP2 Project Plan Summary.*

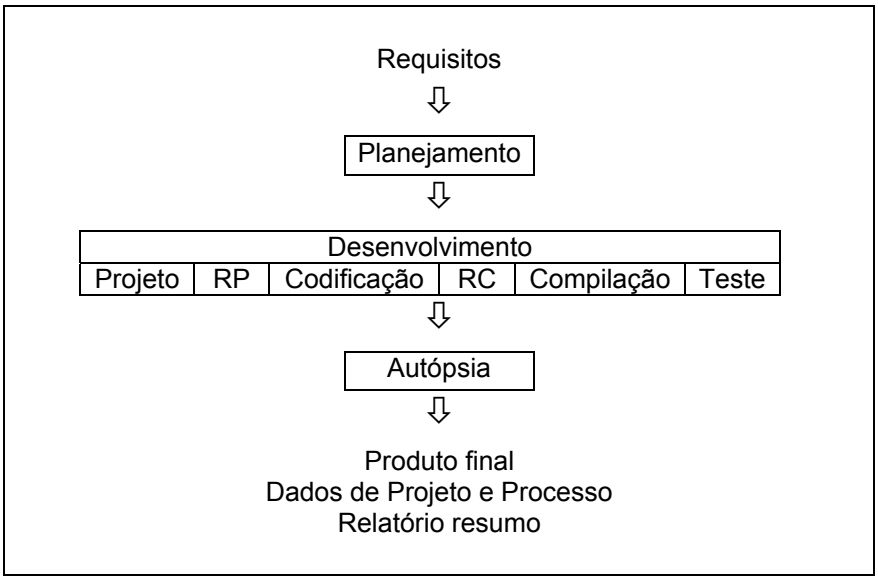

Figura 7. O processo original do PSP2.

Como citado anteriormente, o PSP2.1 introduz padrões para projeto, na forma de quatro *templates*:

- *C66 – Operational Scenario Template*: cenários para prever o funcionamento do programa sob condições normais e anormais;
- *C68 – Functional Specification Template*: especificação das interfaces e lógica dos métodos;
- *C70 – State Specification Template*: especificação dos estados do objeto e suas respectivas transições;
- *C72 – Logic Specification Template*: para a especificação da lógica do programa (utilizando pseudocódigo).

O formulário de planejamento do PSP2.1 incorpora as informações relativas ao custo da qualidade (COQ), passando a se chamar *C63 – PSP2.1 Project Plan Summary.*

3.4 PSP3 – Cyclic Personal Process

O objetivo do PSP3 é viabilizar a aplicação do PSP para projetos maiores, utilizando a estratégia de "dividir para conquistar". Ou seja, ao deparar-se com um projeto maior, o engenheiro de software deve visualizá-lo como várias etapas menores (cerca de 100 a 300 LOC) às quais se pode aplicar ciclos de desenvolvimento do tipo PSP2.1. A idéia principal é produzir um projeto de alto nível[28] e em seguida aplicar ciclos de desenvolvimento utilizando os roteiros e formulários do PSP2.1.

No PSP3 são adicionados mais dois formulários e outros dois recebem alterações. O *C85 – Issue Tracking Log*, funciona como uma espécie de lista de pendências. Ou seja, nele se registram questões que vão surgindo ao longo do desenvolvimento e que necessitam ser tratadas posteriormente. O *C82 – Cycle Summary* tem a função de sumarizar dados dos diversos ciclos de desenvolvimento, tais como: tamanho, tempo por fase e defeitos inseridos e removidos. O *checklist* introduzido no PSP2 para a Revisão de Projeto é complementado com novas informações (especificamente relacionadas aos métodos de verificação de lógica) e passa a chamar-se *C84 – C++ PSP3 Design Review Checklist*. Neste momento, o *checklist* se distancia um pouco da linguagem e se aproxima mais das ferramentas de especificação. Também o formulário de planejamento é alterado para contemplar as informações concernentes ao HDL, passando a chamar-se *C80 – PSP3 Project Plan Summary*.

3.5 Aplicação do PSP na indústria e academia

O PSP vem sendo pesquisado e utilizado em diversas partes do mundo, tanto acadêmica quanto industrialmente. No entanto, os relatos de experiências industriais ainda são poucos e a maioria dos números é proveniente da análise de dados obtidos em cursos de PSP. Os autores deste trabalho não têm conhecimento de relatos de sua utilização industrial no Brasil [REINEHR & BURNETT, 2000].

3.5.1 Os relatos da academia no Brasil

No Brasil, o trabalho de Jones Albuquerque e Silvio Meira, descrito em [ALBUQUERQUE & MEIRA, 1997], propõe adaptações para a utilização do PSP com

[28] Do original, em Inglês: *High-Level Design (HLD)*

a linguagem JAVA, tecendo comparações com Smaltalk. Algumas observações importantes são apresentadas nas conclusões, como: a ênfase quase exclusiva do método nas fases de codificação e testes e a importância de inserir disciplinas como o PSP na formação acadêmica dos engenheiros de software.

O trabalho de Djalma Silva e Paulo Masiero, descrito em [SILVA & MASIERO, 1998], tem o seu foco nas adaptações efetuadas no modelo PSP para ser utilizado com o ambiente de desenvolvimento Powerbuilder, propondo modificações nos formulários e tabelas que são dependentes de linguagem.

Em [SILVA & SPÍNOLA, 1999], Marcos Silva e Mauro Spínola apresentam dados da utilização do PSP academicamente no Brasil, através de um modelo para introdução do PSP. A experiência foi feita usando uma versão customizada, na qual a quantidade de programas de exercício por nível foi ampliada. Foram analisados os dados de 20 alunos de segundo grau, 10 alunos universitários e 2 profissionais desenvolvedores, perfazendo um total de 6.868 LOC em Visual Basic e 14.044 LOC em Pascal. Os resultados apontaram para uma redução média de 25% na densidade total de defeitos dos universitários e profissionais e 36% para os alunos de segundo grau.

3.5.2 Os relatos da academia em outros países

Uma das pioneiras na utilização do PSP em seu currículo foi a Embry-Ridle Aeronautical University (EUA), tendo participado com Watts Humphrey desde o início de sua concepção [HUMPHREY, 1995]. Desde então, universidades de países como: Canadá, Austrália, Alemanha e Itália também incorporaram o PSP aos seus currículos de graduação e pós-graduação [DISNEY & JOHNSON, 1998].

Em [HAYES & OVER, 1997], um dos mais conhecidos relatórios publicados pelo SEI, os autores analisam os dados obtidos em 23 turmas de cursos de PSP, totalizando dados de 298 desenvolvedores. Nesta amostra foram produzidas 300 KLOC e encontrados 22 mil defeitos, em um total de 15 mil horas de desenvolvimento. Foram identificadas melhorias estatisticamente significativas em quatro dimensões: precisão na estimativa de tamanho, precisão na estimativa de esforço, qualidade do produto e qualidade do processo, conforme ilustrado na Tabela 9. No que diz respeito à produtividade, não houve alterações consideradas estatisticamente significativas,

embora se espere que ao se contemplar os testes integrados e testes de sistema ela aumente, devido ao aumento da qualidade dos componentes individuais.

INDICADOR	MELHORIA MÉDIA
Estimativa de Esforço	75%
Estimativa de Tamanho	150%
Defeitos encontrados antes da compilação	50%
Defeitos encontrados em teste unitário	150%

Tabela 9. Melhorias usando o PSP, apresentadas em [HAYES & OVER, 1997].

Disney e Johnson apresentam, em [DISNEY & JOHNSON, 1998], uma série de críticas em relação à qualidade dos dados coletados e analisados pelo PSP. Neste estudo foram avaliadas as informações obtidas em 89 programas desenvolvidos por estudantes da Universidade do Hawaii. Ao todo, foram detectados, analisados, classificados e discutidos 1539 erros. Destes, a grande maioria eram erros de preenchimento dos formulários ou erros de cálculo. Apesar disto, os autores concluem que o PSP é uma ferramenta útil, desde que tomados os devidos cuidados com a coleta e análise dos dados e provida uma ferramenta automatizada.

Em [PRECHELT & UNGER, 2001], os autores relatam os resultados de uma experiência conduzida na Universidade de Karlshure para validar os efeitos do treinamento em PSP. Foram comparados os resultados obtidos por 24 estudantes treinados em PSP com os resultados de 16 estudantes não treinados em PSP, no que diz respeito ao desenvolvimento de uma mesma aplicação. Entre outras conclusões, os autores constatam que o PSP é um método eficiente, porém, apenas o curso no formato padrão não garante que seus participantes utilizarão as técnicas posteriormente.

3.5.3 Os relatos da indústria no Brasil

Não se tem conhecimento de dados sobre a utilização sistemática do PSP na indústria nacional [REINEHR & BURNETT, 2000].

3.5.4 Os relatos da indústria em outros países

Em [FERGUSON et alli, 1997], os autores relatam o caso de três experiências industriais: *Advanced Information Services, Motorola Paging Products Group* e *Union*

Switch & Signal. Em todas elas foram encontradas melhorias referentes ao uso do PSP, quer na precisão das estimativas, quer na redução da densidade de defeitos.

Em [FERGUSON et alli, 1999] Pat Ferguson relata com mais detalhes a experiência da *Advanced Information Services Inc.* no uso de modelos de melhoria de processos em nível organizacional e individual. Ferguson divide a competência da empresa em três diferentes fases: antes de 1992 (onde não era utilizado nenhum modelo de melhoria de processos), entre 1992 e 1995 (com a implantação do modelo SW-CMM) e após 1995 (com o uso do PSP).

Embora a AIS desenvolva sistemas dos mais variados tipos (de *software* comercial a embutido), nas mais variadas plataformas (de *mainframe* a Internet) e nas mais variadas linguagens (de COBOL a JAVA), os dados apresentados na Tabela 10 são para sistemas comerciais. Além destes, verificou-se uma sensível redução na quantidade de defeitos em teste de aceitação dos produtos desenvolvidos por engenheiros de *software* treinados em PSP, bem como um aumento de produtividade. Atualmente, a AIS utiliza internamente um processo que segue os mesmos princípios do TSP [MC ANDREWS, 2000].

INDICADOR	**ATÉ 1992 S/ MODELO**	**1992-1995 CMM**	**APÓS 1995 PSP**
Desvio médio no cronograma	112%	41%	5%
Desvio médio na estimativa de esforço	87%	37%	-4%

Tabela 10. Melhorias na AIS [FERGUSON et alli, 1999].

Em [ESCALA & MORISIO, 1998] e [MORISIO, 1999], os autores discutem a introdução do PSP em uma empresa fabricante de controles numéricos italiana, a FIDIA s.p.a., através de um projeto denominado PSP-NC. A conclusão do trabalho é de que o PSP, assim como qualquer outra ferramenta, pode ser implantado sem adaptações em alguns contextos, mas nem todos. No caso da empresa analisada, várias adaptações tiveram que ser feitas, quer pelos objetivos, quer pela forma como trabalham. Porém, concluem que o PSP é uma ferramenta útil para a melhoria individual.

O *Centre for Software Engineering*, da Universidade de Dublin, apresenta em [CSE, 1998], os dados referentes ao período de treinamento, e em seguida de utilização

profissional, do PSP na *Advent Software*. Este relato constitui um dos estudos de caso do projeto SPIRE[29]. Os números dos três desenvolvedores analisados apontam para uma redução efetiva no esforço despendido na fase de testes: dois dos engenheiros reduziram o tempo de 40% para 15% do total do desenvolvimento. Outro reflexo positivo foi o aumento do tempo despendido na fase de planejamento, ou seja, uma redução proporcional nas fases de maior custo para o projeto.

Em [DELLIEN, 1997], Olof Dellien relata a introdução do PSP na *TXM - Empresa Tecnologica Ericsson*, em Saltillo-Mexico, utilizando um método diferente do proposto por Humphrey. Embora o trabalho não apresente resultados numéricos, diversas considerações são feitas no que diz respeito às adaptações necessárias ao contexto industrial.

3.6 Níveis de Abstração do PSP

No momento de contextualizar o trabalho proposto por esta dissertação, percebeu-se a necessidade de primeiramente representar os diferentes níveis de abstração do modelo PSP, para só então situar e delimitar as alterações que estão sendo propostas. Para esta representação optou-se por utilizar a estrutura de definição de processos apresentada em [MAIDANTCHIK et alli, 1999], na qual os autores apresentam um modelo de processo para equipes geograficamente distribuídas e com grau de maturidade diferenciados, e [ROCHA & MACHADO, 2000], na qual os autores apresentam um modelo de definição de processo para organizações, baseado nos modelos de processos já apresentados no capítulo anterior.

Em ambos os trabalhos é utilizada uma estrutura na qual são consideradas três etapas para se estabelecer um processo que possa ser efetivamente utilizado pela organização:

- Definição do Processo Padrão;
- Especialização do Processo;
- Instanciação para Projetos.

A Figura 8 apresenta estas etapas já sob a ótica do PSP, ou seja, levando em consideração fatores externos que influenciaram sua concepção e os produtos

[29] SPIRE – *Software Process Improvement in Regions of Europe* é um projeto da Comissão Européia para auxiliar a melhoria de processos em pequenas empresas (até 50 funcionários), envolvendo parceiros na Áustria, Itália, Irlanda, Suécia e Reino Unido.

resultantes em cada etapa. Para maiores detalhes das estruturas originais poderão ser consultados os trabalhos de [MAIDANTCHIK et alli, 1999] e [ROCHA & MACHADO, 2000].

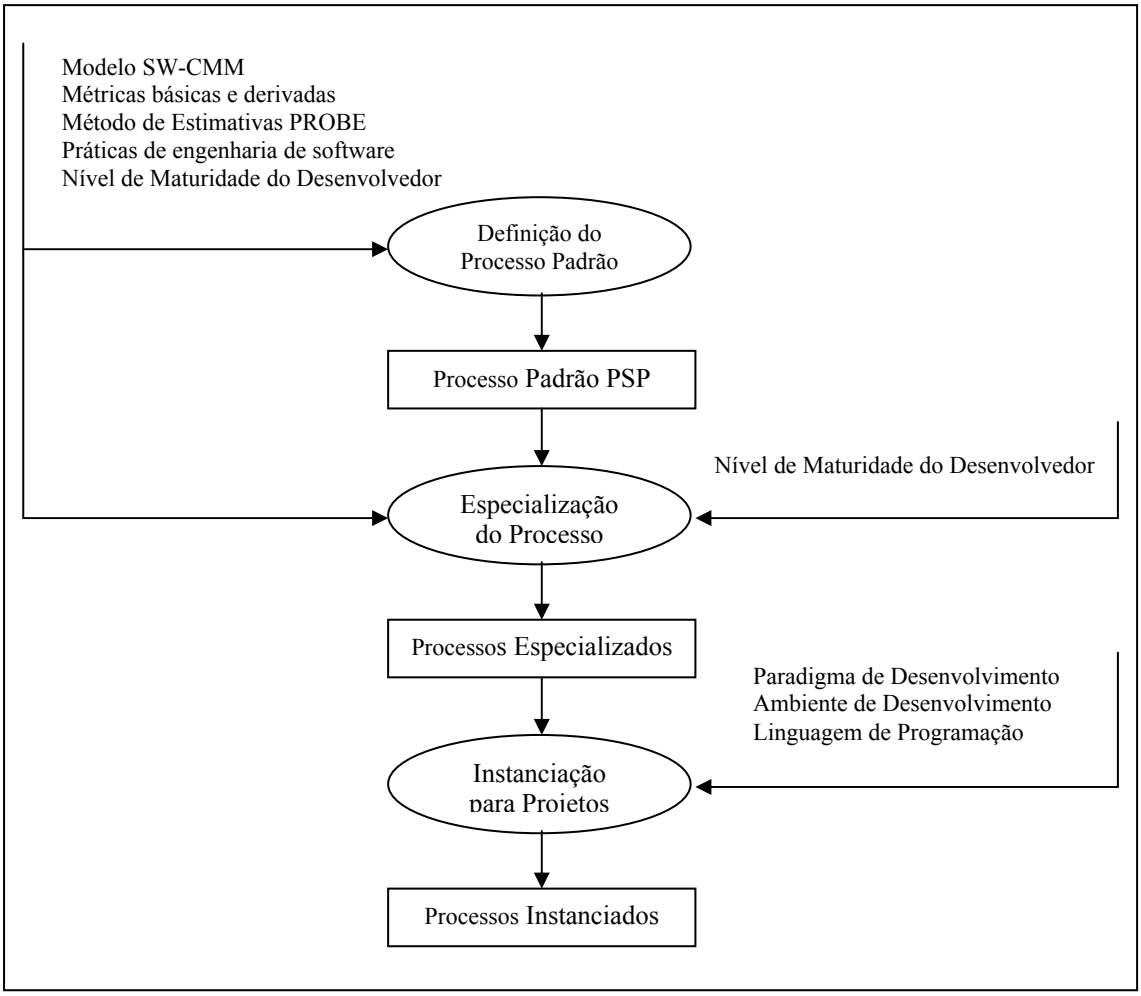

Figura 8. Abstrações do Modelo PSP.

3.6.1 Definição do Processo Padrão

Através do mapeamento realizado, representado na Figura 8, identificou-se que a primeira etapa, Definição do Processo Padrão, no caso do PSP, refere-se à seleção das técnicas e à concepção do modelo propriamente dito, efetuado pelo autor Watts Humphrey. Os fatores de influência deste nível foram:

- Modelo organizacional SW-CMM, do qual foram adaptados para o nível individual os conceitos de maturidade e áreas chave de processo - *KPAs*;

- Métricas básicas: tamanho (LOC), tempo (minutos) e defeitos (quantidade);
- Métricas derivadas: combinação de cada uma das métricas básicas (tamanho, tempo e defeitos) com cada uma das fases (planejamento, projeto, revisão de projeto, codificação, revisão de código, compilação, testes e postmortem), resultando, por exemplo, em: defeitos inseridos e removidos por fase, tempo gasto por fase, defeitos removidos por hora de revisão etc.;
- Método de estimativas PROBE: concebido especialmente para o PSP com base em técnicas estatísticas[30] (conceitos de distribuições, correlação entre duas variáveis, significância, regressão linear etc.) e técnicas de estimativas[31] (*Wideband-Delphi*, lógica *fuzzy*, componente padrão e Análise por Pontos de Função);
- Práticas de engenharia de software: desenvolvimento distribuído em fases claramente definidas (planejamento, projeto, codificação, compilação, testes e autópsia), técnicas de especificação de programa (Cenários, Diagrama de Transição de Estados, Especificação Funcional, Especificação Lógica) e uso extensivo das revisões (de código e de projeto).

O produto resultante da etapa de Definição do Processo Padrão é o Processo Padrão PSP, que pode ser considerado como sendo o processo completo PSP3.

3.6.2 Especialização do Processo

Fazendo uma analogia com [MAIDANTCHIK et alli, 1999], na qual os autores definem uma estrutura de processo para atender equipes geograficamente dispersas, podemos perceber que nesta etapa, Especialização do Processo, o Processo Padrão PSP é adaptado de acordo com o nível de maturidade do desenvolvedor. Ou seja, dependendo do grau de conhecimento e aplicação do modelo, o desenvolvedor utilizará um dos Processos Especializados (PSP0, PSP0.1 etc.). Cada um destes níveis nada mais é do que o Processo Padrão PSP (PSP3) subtraído de atividades de acordo com o estágio no qual se encontra o desenvolvedor.

[30] O Apêndice A de [HUMPHREY, 1995] dedica-se a explicar as bases estatísticas utilizadas na concepção do PROBE.
[31] O capítulo 5 de [HUMPHREY, 1995] descreve sucintamente os métodos de estimativas utilizados na concepção do PROBE.

A saída da etapa de Especialização do Processo é o conjunto de Processos Especializados de acordo com a maturidade do desenvolvedor: PSP0, PSP0.1, PSP1, PSP1.1, PSP2, PSP2.1 e o próprio PSP3.

3.6.3 Instanciação para Projetos

Nesta etapa, Instanciação para Projetos, o Processo Especializado necessita ser novamente refinado, levando-se em conta as características próprias do projeto, como o paradigma de desenvolvimento, o ambiente de desenvolvimento e a linguagem de programação.

- Paradigma de Desenvolvimento: as ferramentas utilizadas na especificação dos programas, introduzidas no nível PSP2.1 na forma de *templates*, são influenciadas pelo paradigma de desenvolvimento. Isto é claro quando se analisa, tanto a terminologia do paradigma orientado a objetos adotada (classes, objetos e métodos), quanto a própria estrutura dos formulários de especificação (agregando atributos e métodos em forma de classes).
- Ambiente de Desenvolvimento: o ambiente utilizado para o desenvolvimento influencia sobremaneira a instanciação para projetos, conforme os argumentos de [SILVA & MASIERO, 1998] com relação ao ambiente Powerbuilder, o qual compila automaticamente os programas à medida que o desenvolvedor fecha o programa. Também a utilização de ferramentas gráficas CASE na definição de programas pode levar à eliminação de alguns dos *templates* puramente orientados a caracter do PSP.
- Linguagem de Programação: um exemplo da influência da linguagem de programação pode ser visto nas tabelas de categorias de tamanho relativo de objetos utilizadas no método de estimativa de tamanho PROBE. Essas categorias e tamanhos serão variáveis de acordo com a linguagem que está sendo empregada. Segundo o próprio Humphrey em [HUMPHREY, 1995], os dados utilizados com o propósito de estimativas devem refletir precisamente a linguagem e as práticas que estão sendo adotadas. Para isto, ele apresenta categorias e tamanhos para C++ e para Object Pascal em [HUMPHREY, 1995]. Outras atividades previstas no PSP também sofrem influência da linguagem adotada: padrão de codificação, padrão de contagem

de LOC, os *checklists* para a revisão de código e os *checklists* para revisão de projetos. Trabalhos como os de [ALBUQUERQUE & MEIRA, 1997] e [SILVA & MASIERO, 1998], refletem especializações neste nível, quando consideram ambientes JAVA e Powerbuilder, respectivamente.

A saída da etapa de Instanciação para Projetos é o conjunto de Processos Instanciados para os diferentes contextos de projeto do desenvolvedor.

3.7 Análise do PSP

De uma maneira geral, os analistas e programadores desenvolvem sistemas utilizando as práticas correntes da organização onde trabalham ou aquelas aprendidas com seus pares. Porém, nem sempre estas práticas são as melhores ou levam aos melhores resultados. O PSP é um método que oferece, principalmente, uma disciplina pessoal ao desenvolvedor, introduzindo e incentivando o uso das melhores práticas de engenharia de software. O fato de utilizar processos bem definidos e documentados na forma de roteiros traz ao desenvolvedor segurança de saber exatamente qual é o próximo passo a seguir.

Conforme apontado em todos os estudos apresentados neste capítulo, o PSP promove melhorias na capacidade de estimativa dos desenvolvedores, bem como reduz a densidade de defeitos. Além disto, propicia ao engenheiro de *software* conhecer quantitativamente a forma como trabalha, ou seja, o desempenho do seu processo pessoal de desenvolvimento, possibilitando identificar e promover as melhorias necessárias.

O que se observa na prática é que, para que o PSP possa ser realmente utilizado em empresas, e consequentemente comprovar os resultados apregoados, é necessário solucionar algumas questões importantes que serão discutidas nas próximas seções.

3.7.1 O PSP e o primeiro "P": o processo

Conforme dito anteriormente, o PSP oferece um processo bem definido e mensurável, detalhado na forma de roteiros e formulários, e escalonado em níveis evolutivos de maturidade. Não é tão genérico e abstrato quanto os modelos e normas com foco organizacional, o que já oferece uma base mais paupável para o início de sua utilização.

O elemento central do PSP é o processo de estimativas PROBE, que oferece meios para que o desenvolvedor melhore a sua capacidade de estimativas e, em última instância, sua capacidade de planejamento. No entanto, apesar de oferecer algumas orientações para a montagem da base de categorias de tamanho, essa etapa não faz parte do processo PSP e nem do método de estimativas PROBE, conforme detalhado na próxima seção. Devido à sua importância, esse ponto deveria fazer parte integrante do método. Além disso, deixa uma lacuna importante no processo de estimativas, sujeita a um elevado grau de subjetividade, que é a seleção das categorias de tamanho dos *proxies*.

3.7.2 O PSP e o segundo "P": o projeto

Considerado na dimensão de processo, o PSP é um *framework* genérico e independente de tecnologia. Porém, no momento de sua instanciação para utilização prática em projetos, ele passa a ser dependente do ambiente no qual está sendo introduzido. Isso é claramente demonstrado nos trabalhos de [ALBUQUERQUE & MEIRA, 1997] e [SILVA & MASIERO, 1998], que adaptaram diversos formulários do método para uso com diferentes linguagens de programação (JAVA e Powerbuilder, respectivamente). Esta dependência é bem explorada em [ALBUQUERQUE & MEIRA, 1997] ao comparar os resultados obtidos na programação JAVA e Smaltalk.

Conforme abordado na subseção anterior, o método para a montagem desta base inicial de estimativas, que é o elemento central do método de estimativas PROBE, e, consequentemente o elemento central do próprio PSP, apesar de discutido em [HUMPHREY, 1995], não se encontra estruturado na forma de processo e não faz parte dos roteiros e formulários. O próprio PROBE parte do princípio que o desenvolvedor deverá julgar o tamanho relativo do objeto e estimá-lo em termos de LOC/método quando for aplicar a um projeto. Esta atividade envolve um certo grau de subjetividade que pode comprometer todo o processo de planejamento do projeto. Um dos objetivos desta dissertação é reduzir essa subjetividade, oferecendo orientações mais detalhadas para a concepção da estimativa.

Outro ponto importante é o que concerne à coleta e análise dos dados. O PSP é um processo complicado e trabalhoso, com inúmeros formulários a serem preenchidos, e sem suporte automatizado para utilização em projetos, além de mais sujeito a erros

[DISNEY & JOHNSON, 1998], causa uma sobrecarga de tarefas ao engenheiro de software que pode definitivamente afastá-lo da disciplina.

A ferramenta PSP2000, proposta e descrita pelos autores em [WENCESLAU et alli, 2001], apesar de já representar um avanço para apoiar a utilização do PSP em projetos, ainda deixa lacunas importantes na automatização, como as orientações para a estimativa e a montagem da base de categorias.

3.7.3 O PSP e o terceiro "P": a pessoa

O PSP foi desenvolvido para ser utilizado como uma ferramenta de desenvolvimento profissional pessoal. Ou seja, ela foi projetada para ser usada pelo indivíduo, coletando e analisando suas próprias métricas. No entanto, as pessoas não trabalham de forma isolada e em geral, são agrupadas em forma de equipes para compor um projeto. Nesse momento, surge a necessidade de se trabalhar as métricas individuais de forma coletiva, em prol do projeto.

Surge, então, um aspecto de ordem psicológica muito importante no PSP: se o profissional não se sentir seguro de que os seus dados pessoais não serão utilizados com o propósito de avaliação, sua tendência será a de mascarar os dados reais, tornando-os convenientes a cada situação e deturpando a finalidade da coleta.

Esta questão envolve a necessidade da empresa encontrar meios de assegurar a privacidade de dados individuais, sem, contudo prejudicar as necessidades de mensuração existentes nos projetos.

3.7.4 O PSP e o quarto "P": o problema

Apesar de ter sido concebido como um processo genérico que cobre todas as fases do ciclo de vida de software, o PSP tem o seu foco nas fases de codificação e testes, dando pouca ênfase às fases de captura e análise de requisitos [ALBUQUERQUE et alli, 1998] e [SILVA & MASIERO, 1998]. Esta ênfase é referenciada pelo próprio Humphrey quando afirma que o PSP, conforme descrito em [HUMPHREY, 1995], se concentra nas fases de projeto detalhado, codificação e testes. Uma das razões por ele apresentadas é a de que essas fases se adaptam mais facilmente a exercícios em sala de aula. Porém, uma outra razão não explorada é que as melhorias identificadas são mais facilmente perceptíveis na camada de projeto físico, uma vez que

os requisitos não são mais tão vagos e o projeto conceitual pode ser concebido de forma mais precisa. Isso influencia sobremaneira a percepção da melhoria na capacidade de estimativas do desenvolvedor.

Conforme ilustrado na Figura 5, apresentada no início deste capítulo, a primeira fase do PSP0, ou seja, a fase de planejamento, parte do princípio de que se tem um documento de requisitos. Na prática isto não é realidade porque o trabalho do desenvolvedor inicia com uma vaga declaração do usuário sobre suas necessidades.

Na mesma figura podemos perceber ainda que, ao entrar no que o método chama de fase de desenvolvimento, não existe separação entre análise e projeto. Porém, na verdade, os objetivos de cada uma destas etapas são distintos e, não raro, elas podem inclusive ser desenvolvidas por pessoas diferentes. Esta visão é compartilhada pela maioria das metodologias de desenvolvimento.

Em [JACOBSON et alli, 1999], temos:

> "Ao executar a análise separadamente, ao invés de executá-la como parte integrante do projeto e implementação, podemos analisar uma parte maior do sistema, com pouco custo (...) A análise provê uma visão geral do sistema que pode ser difícil de obter estudando os resultados do projeto ou implementação uma vez que muitos detalhes foram introduzidos."

Isto nos levar a perceber que existe uma brecha no PSP no que diz respeito ao quarto "P", *problema*. Ele não endereça adequadamente esta questão, uma vez que foca as fases de projeto e construção, em detrimento da fase de descoberta e análise de requisitos.

3.7.5 O PSP e o quinto "P": o produto

O PSP demonstra uma preocupação constante com a qualidade do produto de software, introduzindo, desde o primeiro nível, através do PSP0, a mensuração e a classificação dos defeitos inseridos e removidos em cada fase. Além disso, preocupa-se também com o custo dessa qualidade, incentivando o desenvolvedor a descobrir e remover os defeitos o mais cedo possível no ciclo de vida.

No terceiro nível, através do processo PSP2, o passo mais efetivo rumo à melhoria da qualidade é dado ao inserir os conceitos de revisão de código e revisão de projeto detalhado. E, posteriormente, no PSP2.1, outro passo significativo é dado ao incentivar e apoiar o desenvolvimento e a documentação do projeto detalhado.

3.7.6 Outros aspectos importantes

O treinamento, conforme concebido por Watts Humphrey, é longo e oneroso para as empresas, cerca de 150 horas/engenheiro de software, o que equivale a mais de um terço do tempo médio de uma pós-graduação *latu-sensu*. Em parte, isto é devido ao fato de que, para convencer os profissionais do custo x benefício do método, Humphrey introduz passo a passo cada uma das etapas através de exercícios pré-definidos e da coleta/análise dos dados. Este tipo de custo, de forma geral, é difícil de ser absorvido pelas organizações que acabam focando os seus recursos em treinamentos relacionados à tecnologia propriamente dita

E, finalmente, as empresas possuem os seus próprios processos e padrões, formais ou não, de desenvolvimento e manutenção de software e, apesar de não ser o escopo do PSP o envolvimento de toda a organização, ele exige uma grande mudança de cultura, ao menos em nível individual e de equipes. Por isso, o ambiente ideal para o PSP é aquele onde a empresa apóia, promove e incentiva a melhoria individual e organizacional, preferencialmente rumo a um modelo organizacional de melhoria de processos, conforme relatado em [FERGUSON et alli, 1999].

3.8 Considerações sobre o capítulo

Este capítulo apresentou em detalhes o modelo de melhoria de processos individual PSP, conceituando os seus níveis de abstração. Além disso, dedicou-se a explorar a sua utilização no Brasil e em outros países, tanto acadêmica, quanto industrialmente.

Ao analisá-lo em relação aos cinco ”P”s que influenciam o desenvolvimento de software: processo, projeto, pessoas, problema e produto, apontou as lacunas que precisam ser preenchidas no momento de instanciação do processo básico para sua aplicação no ambiente de projetos das organizações.

CAPÍTULO 4 - PSPi - A INSTÂNCIA PROPOSTA

É consenso que as empresas e os desenvolvedores que promovem melhorias em seus processos de software tendem a promover melhorias, tanto na qualidade do produto final, quanto no desempenho de seus projetos [PAULK et alli, 1994], conforme apresentado nos capítulos anteriores. No entanto, processos genericamente concebidos em modelos como os vistos no capítulo 2 (ex.: a família CMM, a NBR ISO/IEC 12207, e até mesmo o PSP) não podem ser utilizados diretamente nas organizações sem que adaptações nos próprios processos e métodos tenham que ser promovidas, aderindo-os aos ambientes nos quais serão inseridos.

Segundo Jacobson, em [JACOBSON et alli, 1999]:

> "Um processo de desenvolvimento de software único não pode ser aplicado em todos os lugares! Os processos variam porque existem em diferentes contextos, desenvolvem diferentes tipos de sistemas e atendem a diferentes tipos de restrições de negócios (Ex.: cronograma, custo, qualidade e confiabilidade)."

Ou, ainda, em [ROBERTSON & ROBERTSON, 1999]:

> "... nós sabemos, por experiência, que cada projeto precisa de um processo diferente pelo simples motivo que cada projeto é diferente."

Na prática, tanto as organizações, como os desenvolvedores, ainda sentem grande dificuldade em promover estas adaptações. Prova disto está nas próprias normas estabelecidas pelos organismos de padronização, como a NBR ISO/IEC 12207 que, além de possuir um processo específico para este fim, denominado Processo de Adaptação, possui ainda um guia [ISO/IEC, 1998c] para facilitar a sua customização e aplicação. Para manterem-se suficientemente genéricas para atender aos diversos contextos a que se destinam, estas normas não prescrevem detalhes de instanciação, como por exemplo, o uso de determinado modelo de ciclo de vida ou técnica de engenharia de software.

Conforme ilustrado na Figura 5 (pág. 40), e já discutido nos capítulos anteriores, o PSP foi concebido para ser um processo genérico, adaptável às mais variadas necessidades dos desenvolvedores. No entanto, no momento da sua aplicação prática, ou

seja, na sua instanciação para ser utilizado pelo engenheiro de software, ele passa a não ser mais genérico, uma vez que baseia-se em premissas dependentes de paradigma de desenvolvimento e linguagem de programação, conforme explorado na subseção 3.6.3 (pág. 54) e ressaltado na Figura 9.

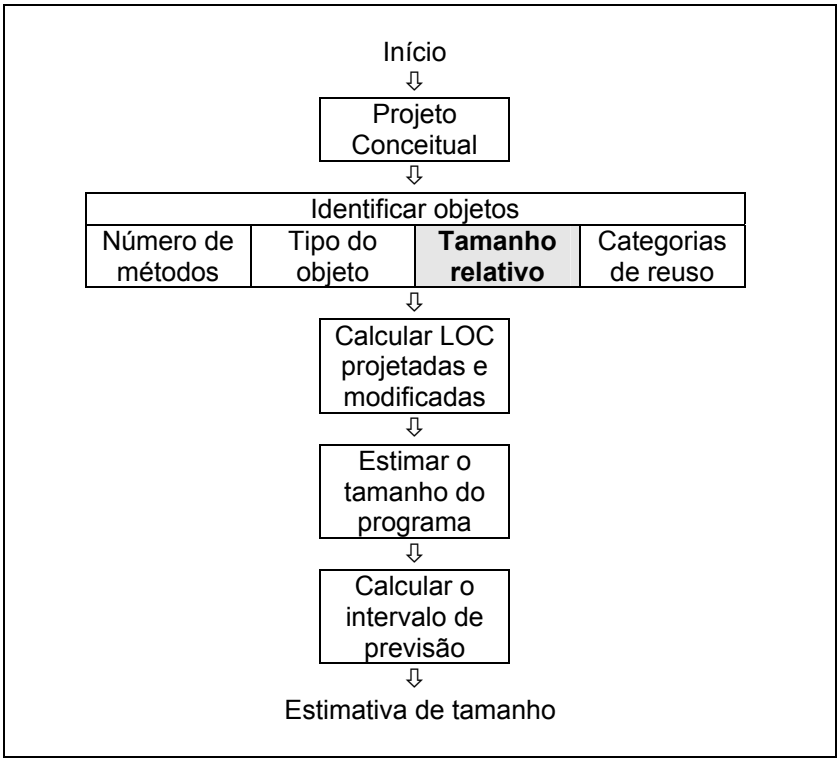

Figura 9. Método de estimativas PROBE – tamanho relativo [HUMPHREY, 1995]

No momento em que o desenvolvedor precisa adaptar o processo para a sua realidade, é que surgem as dificuldades:

- Como montar uma base de categorias de tamanho relativo (Figura 9) aderente ao seu ambiente de desenvolvimento?
- Como carregar essa base com informações se ainda não há nenhum dado pessoal de projeto anterior?
- Como avaliar se uma base de categorias está adequada sem ter que dominar todas as ferramentas estatísticas que embasam o PSP?
- Como reduzir o fator subjetivo existente no momento de optar pela categoria de tamanho mais adequada?

O PSPi, instância do PSP proposta por esta dissertação, visa auxiliar as organizações que possuem ambiente de sistemas legados de grande porte, na tarefa de endereçar essas questões, preparando-as para, posteriormente, iniciar o processo de melhoria organizacional, utilizando o modelo CMM-I ou similar.

Neste capítulo será apresentada a visão geral das complementações ao modelo PSP propostas por esta dissertação. Serão apresentados, ainda, os formulários e procedimentos do PSP que foram adaptados e os formulários e procedimentos que foram criados, como o processo *PSPi0.2* criado para a montagem da base de categorias de tamanho. Nos capítulos seguintes são descritos: a ferramenta construída para automatizar o processo e os resultados obtidos com a aplicação da base montada. Como as modificações propostas nesta dissertação situam-se no terceiro e último nível de abstração do PSP, Instanciação para Projetos, descrito no capítulo 3, subseção 3.6.3 (pág. 54), sua relação com o estudo de caso é bastante estreita e às vezes, ambos chegam a se confundir, como se poderá observar nos próximos capítulos.

A seção 4.1 desse capítulo apresenta o ambiente legado corporativo e suas características; a seção 4.2 oferece uma visão geral da instância PSPi, proposta por esta dissertação e as seções seguintes descrevem com detalhes cada um dos elementos que compõem a instância PSPi.

4.1 O ambiente legado corporativo

Na maioria das organizações que possuem sistemas legados no ambiente de grande porte (*mainframe*), a linguagem padrão de terceira geração utilizada ainda é o COBOL[32]. Evidências desse fato puderam ser observadas por ocasião da preparação para enfrentar o "*bug* do milênio", onde a demanda por desenvolvedores COBOL cresceu inesperadamente no mercado nacional e internacional.

Isto nos remete a pensar na dicotomia levantada por Booch, apresentada na introdução desta dissertação, a respeito da necessidade de manter e evoluir os sistemas legados e adotar as novas tecnologias emergentes. Para essas organizações, continuar cumprindo seus objetivos de negócio, muitas vezes, significa continuar mantendo e evoluindo seus sistemas desenvolvidos em COBOL e outras linguagens de terceira

[32] COBOL (COmmon Business Oriented Language).

geração, e passar a oferecer interfaces com o usuário desenvolvidas utilizando as novas tecnologias, linguagens e paradigmas.

Um exemplo dessa abordagem é o sistema financeiro nacional que, em geral, ainda possui os seus sistemas principais centralizados, processando de forma *batch*, em ambiente *mainframe*. Porém, seus produtos e serviços são oferecidos aos clientes através de interfaces desenvolvidas utilizando as novas tecnologias disponíveis como: Internet, WAP, centrais de atendimento automatizadas etc.

Em [SEACORD et alli, 2001], ao discutir estratégias para a modernização de sistemas legados, os autores utilizam como estudo de caso um sistema de uma rede varejista composto de aproximadamente três milhões de linhas de código COBOL. Esforços como esse, além de onerosos e repletos de riscos, chegam a levar até seis anos para serem concluídos. Durante esse período, as atividades de manutenção continuam em andamento e continuarão, até que toda a modernização esteja concluída e o sistema possa ser substituído.

Como a instância que está sendo proposta, PSPi, se insere nesse contexto, o dos sistemas legados corporativos, todas as alterações que dependem de linguagem estarão considerando a linguagem COBOL. Com essa abordagem, o trabalho aqui desenvolvido poderá ser utilizado de forma abrangente nessas empresas, cobrindo a maior parte do seu ambiente *batch* legado.

A parte on-line do ambiente legado corporativo não será tratada por este trabalho devido à sua diversidade. Em geral, essa porção do ambiente é composta por linguagens de quarta geração heterogêneas e dependentes do Sistema Gerenciador de Banco de Dados em uso na organização (ex.: linguagem NATURAL para ambientes ADABAS, linguagem CA-IDEAL para ambientes CA-DATACOM etc.). Isso não significa que essa proposta não possa ser por eles utilizada, mas, sim, que ter-se-á que desenvolver padrões (de codificação, de contagem de LOC e de revisões) e orientações (de codificação e de estimativa) específicos para cada linguagem. Novamente, isto se deve ao fato que a abordagem desta dissertação concentra-se no terceiro nível de abstração do PSP, Instanciação para Projetos, e que esse nível é extremamente dependente do ambiente utilizado.

4.2 Visão Geral do PSPi

A Figura 10 oferece uma visão geral das adaptações que se fizeram necessárias ao PSP para compor a instância proposta PSPi. O retângulo mais interno da figura representa o PSP original e o mais externo, as adaptações desenvolvidas por esta dissertação e denominadas, em seu conjunto, PSPi.

É importante observar que as adaptações que estão sendo propostas por esta dissertação diferem substancialmente, em abrangência e profundidade, das demais abordagens de outros trabalhos no Brasil, uma vez que afetam inclusive o elemento central do PSP, o método de estimativas PROBE.

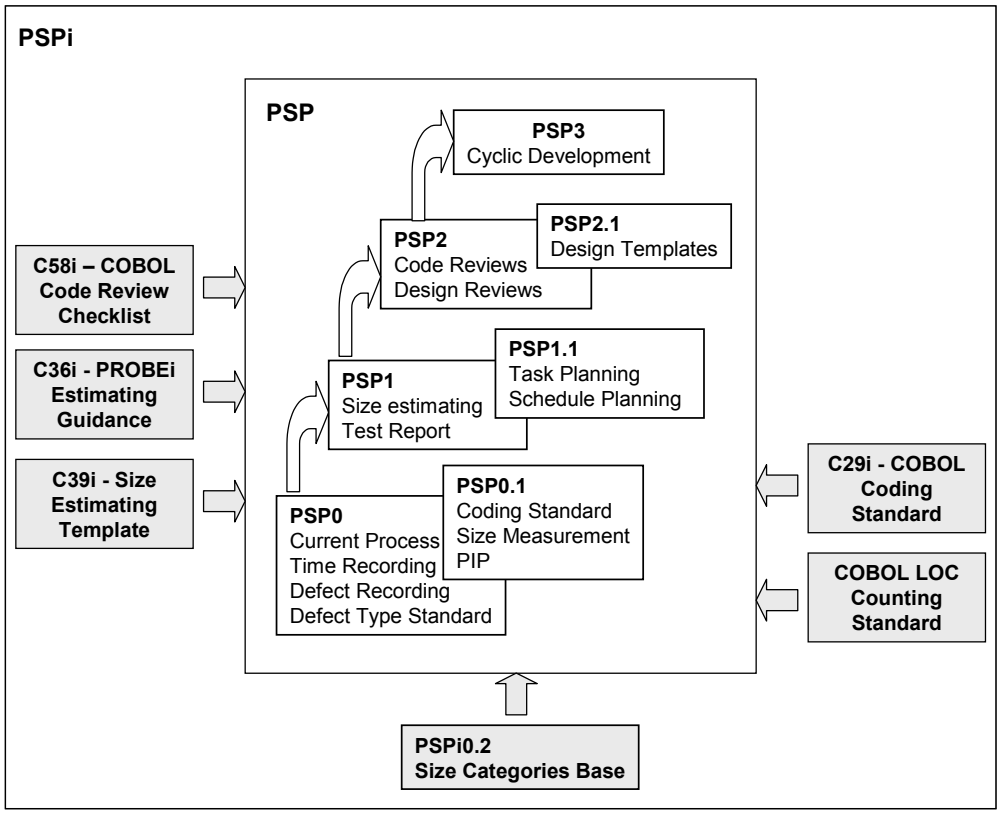

Figura 10. Instância proposta PSPi - Visão Geral.

Conforme se pode observar na Figura 10, os elementos que foram inseridos e/ou alterados para compor a instância PSPi, e que serão detalhados nas próximas subseções, são:

- Dois novos padrões: um para a codificação usando a linguagem COBOL, *C29i – COBOL Coding Standard* e outro para a contagem das LOC, COBOL LOC *Counting Standard*;
- Um novo *checklist*: para apoiar a revisão de código de programas COBOL, *C58i - COBOL Code Review Checklist;*
- Um novo processo: para auxiliar a montagem da base de categorias de tamanho relativo, *PSPi0.2 – Size Categories Base*;
- Uma extensão ao método PROBE: um guia para auxiliar a estimativa de programas COBOL, C*36i - PROBEi Size Estimating Guidance*; e
- Uma alteração de formulário: gerando o formulário alterado *C39i – Size Estimating Template*, de modo a torná-lo aderente ao guia de estimativas proposto.

Com o objetivo de manter a compatibilidade com o método PSP original, que não possui tradução oficial para o português, optou-se por não traduzir os roteiros, formulários e padrões, nem do método original, nem da instância proposta PSPi. Por isso, ao longo desse capítulo alguns elementos, como tabelas e formulários, aparecem em inglês, denotando os elementos que foram desenvolvidos e incorporados ao PSP original através da instância PSPi.

No material original do PSP, os roteiros, padrões e formulários que fazem parte do Anexo C de [HUMPHREY, 1995] são identificados pela inicial "C", como por exemplo o padrão para codificação usando C++, denominado *C29 – C++ Coding Standard*. Já os elementos que se encontram descritos apenas no decorrer do texto de [HUMPHREY, 1995] não possuem a letra "C" no início, como, por exemplo, o padrão de contagem de LOC de C++, denominado *C++ Counting Standard*. Novamente, com o objetivo de manter a compatibilidade com o PSP original, os elementos propostos por esta dissertação, quando possível, seguem a nomenclatura e a numeração originais, seguidas da letra "i", representando a instância que está sendo proposta. Por exemplo, o padrão para codificação COBOL, que substitui o padrão para C++ do PSP, recebeu o nome de *C29i – COBOL Coding Standard*.

As próximas subseções detalham as complementações propostas por esta dissertação, através da instância PSPi.

4.3 *C29i - COBOL Coding Standard*

A linguagem COBOL, conforme ilustrado na Tabela 11, apresenta uma estrutura relativamente rígida em relação à codificação, forçando o desenvolvedor a inserir as informações em áreas preestabelecidas do programa (*divisions* e *sections*[33]*)*.

Ao contrário de outros tipos de linguagem, cuja estrutura é mais flexível, o COBOL segrega as informações dentro do programa de acordo com a sua finalidade, separando as definições relativas: à identificação do programa na *Identification Division*, ao ambiente de processamento na *Environment Division*, à definição dos dados na *Data Division* e ao código propriamente dito na *Procedure Division*.

ESTRUTURA	**FINALIDADE**
IDENTIFICATION DIVISION	Área reservada à identificação do programa.
ENVIRONMENT DIVISION	Área reservada à identificação do ambiente de execução do programa e às especificações externas dos arquivos.
CONFIGURATION SECTION	Seção reservada à identificação do ambiente de execução do programa.
INPUT-OUTPUT SECTION	Seção reservada às especificações externas dos arquivos.
DATA DIVISION	Área reservada à identificação dos dados utilizados pelo programa (especificações internas de arquivos, visões de bancos de dados e variáveis de memória)
FILE SECTION	Seção reservada às especificações internas dos arquivos (registros).
WORKING STORAGE SECTION	Seção reservada à declaração das variáveis de memória e das visões das tabelas de banco de dados.
PROCEDURE DIVISION	Área reservada à codificação das instruções de programa (comandos da linguagem).

Tabela 11. Estrutura da Linguagem COBOL.

[33] Optou-se por manter a nomenclatura original da linguagem COBOL (*divisions, sections etc.)*, uma vez que esses termos são utilizados normalmente nas empresas usuárias da linguagem no Brasil.

A liberdade para o codificador restringe-se a modularizar, ou não, a codificação, utilizando o recurso de dividir a *Procedure Division* em *sections.* Isto facilita o trabalho de definição do padrão de codificação e também do padrão para contagem de LOC da linguagem COBOL.

O padrão para a codificação que está sendo proposto por esta dissertação encontra-se descrito na Tabela 12. Esse padrão foi baseado no original, *C29 – C++ Coding Standard*, ao qual foram acrescentadas as particularidades da linguagem COBOL.

Nesse padrão buscou-se enfatizar dois aspectos que conferem facilidade de manutenção ao programa:

- Documentação: através da inserção de uma área inicial destinada a comentários padronizados com informações sobre a função do programa, o codificador, as alterações etc. e do incentivo ao uso extensivo de comentários explicativos no corpo do programa.
- Clareza: através da definição de padrões para o nome de variáveis, incentivo ao uso de linhas em branco para separar blocos principais e incentivos à programação modularizada (baixo acoplamento e alta coesão).

PROGRAM AREA	TYPE	DESCRIPTION
IDENTIFICATION DIVISION	Header insertion	• Insert a descriptive header into the IDENTIFICATION DIVISION, just after the PROGRAM-ID clause. • Keep it updated, as new features are added, previous ones are modified or deleted.

	Header Format	`*****************************************************`
		`* APLICATION.....the name of the application to which *`
		`*                the program belongs                  *`
		`* ANALIST........the name of who had defined the      *`
		`*                program                              *`
		`* PROGRAMMER.....the name of who is coding the        *`
		`*                program                              *`
		`* DATE..........the date program development started *`
		`*                                                     *`
		`* DESCRIPTION....a short description of program´s     *`
		`*                function                             *`
		`* MODIFICATIONS..a short description of updates       *`
		`*                including: date, author and a short  *`
		`*                description                          *`
		`*-----------------------------------------------------*`
		`* DATE      AUTHOR     DESCRIPTION                    *`
		`* 99/99/99  XXXXXXXXX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX *`
		`* ...                                                 *`
		`* ...                                                 *`
		`*****************************************************`

WORKING STORAGE SECTION	Identifiers Names	• Use descriptive names for all variables, sections, constants and other identifiers. Avoid single-letter variables.
		• Use the following prefixes to indicate the intended use of the identifier:
		  ◦ "as": indicating a working variable
		  ◦ "tb": indicating a table (or an array)
		  ◦ "ac": indicating cumulative variable
		  ◦ "msg": indicating constants to be used in error messages
		  ◦ "ch": indicating variables to be used as key controls or flags
		  ◦ "ind": indicating variables to be used as table (array) indexes

	Name example	• Well defined names: as-client-name, tb-product-categories, ac-balance, msg-read-error, ch-client-ok, ind-product-categories • Bad defined names: client, x, ind, as.
	Grouping variables	• Group similar variables under a single "01" level declaration, separating them by their purpose.
	Grouping examples	• 01 AS-STORAGE-AREAS. • 01 AS-COUNTERS. • 01 AS-INTERNAL-TABLES.
PROCEDURE DIVISION	General	• Code the program in order to provide readability, maintainability and good performance.
	Main section	• Code a main section, with very few LOC, in order to call the *starting section, the processing section and the normal ending section.* • Include here the ending command GOBACK.
	Starting section	• Code all needed initializations in a separate section called *starting section.*
	Normal ending section	• Code all needed commands to perform a normal ending exit of the program, called *normal ending section.*

PROGRAM AREA	TYPE	DESCRIPTION
	Abnormal ending section	• Code all needed commands to perform an abnormal ending exit of the program, called *abnormal ending section.*
	Processing section	• A section that holds the main logical processing, calling the other sections stated bellow.
	VSAM and sequential files	• Provide one separated section to each operation to be made in a VSAM or a sequential file: open, close, read and write. • Just after each operation, code the proper status test and error handling.
	Database tables	• Provide one separate section to each operation to be made in a database table: cursor declaration (usually included in the *Working Storage Section)*, cursor open, cursor close, select, insert, update, and delete. • Just after each operation, code the proper SQL returns status test and error handling.
	External Calls	• Provide one separated section to each subroutine call. • Just after each external call, code the proper status test and error handling.
	Report controls	• Code all needed commands to control reports in a separated section
	Other business rules sections	• Divide the program into small and understandable sections, searching for high cohesion and low coupling. • Each section should perform one, and only one, function (high cohesion). • One section should not refer to another section that is at the same level (low coupling). • End each major section by the keyword EXIT, placed on its own line.
DOCUMENTATION	Comment	• Document the code so that the reader can understand its operation. • Precede a major program section by a block of comments that describes the processing that is done inside the section. • Comments should explain both the purpose and the behavior of the code. • Avoid using obvious comments that do not clarify the code.

	Comment example	`*****************************************************`
		`* Examines the contents of the array "grades" and *`
		`* calculates the average grade for the class *`
		`*****************************************************`

	Blank spaces	• Write programs with sufficient spacing so that they do not appear crowded. • Separate every major block of commands with a blank line.
	Indenting	• Indent every level of nested commands, leaving 4 blank spaces for each indented level.

	Indenting example	`IF IND-TAB-REM GREATER 0 THEN`
		`  PERFORM 021-PROCESS-BALANCE`
		`ELSE`
		`  PERFORM 022-PROCESS-REJECTION`
		`END-IF.`

	Skipping pages	Precede each DIVISION and each major SECTION of the program by a slash (/). This will place the DIVISION or SECTION into a new page in compile result printing.

Tabela 12. ***C29i - COBOL Coding Standard.***

4.4 COBOL LOC Counting *Standard*

Estabelecer um padrão adequado para a contagem de linhas de código é muito importante para garantir que a medição possa ser reproduzida com os mesmos resultados [HUMPHREY, 1995]. Conforme visto no capítulo 3, nos trabalhos de adaptação do PSP apresentados, o padrão para a contagem de LOC é outro elemento bastante aderente às condições do ambiente de desenvolvimento e da própria linguagem de programação.

O estabelecimento desse padrão para a linguagem COBOL acaba sendo simplificado pelas razões já apresentadas anteriormente:

- A linguagem possui uma estrutura relativamente rigorosa de codificação, delimitando claramente onde as informações devem estar codificadas, conforme demonstrado na Tabela 11;
- Menor possibilidade de aninhar comandos em uma mesma linha de codificação;
- Não há como misturar a definição de elementos de dados com a codificação propriamente dita.

O padrão para a contagem das LOC que está sendo proposto por esta dissertação, descrito na Tabela 13, considera como LOC apenas as linhas úteis da *Procedure Division.* As linhas das demais áreas do programa não devem ser contadas como LOC.

Como se pode observar, optou-se por seguir a ordem da estrutura COBOL que já havia sido apresentada na Tabela 11: *Identification Division, Environment Division, Data Division e Procedure Division*.

Considerou-se como linha útil qualquer linha compreendida entre a sentença reservada *Procedure Division*, incluindo essa sentença, e o final do código. Foram desconsideradas, para efeito de linhas úteis, as linhas em branco, as linhas de comentários e as linhas que contém apenas o ponto de finalização (.)[34].

[34] A linguagem COBOL utiliza o caracter de ponto (.) para terminar sentenças e alguns comandos. Alguns codificadores têm o hábito de codificar o ponto em uma linha separada, como forma de aumentar a clareza do programa. Para o padrão proposto, essa linha será considerada como uma linha de comentário.

Os padrões estabelecidos facilitam a contagem automatizada das linhas de código, uma vez que basta contar as linhas físicas do código e desprezar aquelas que não são consideradas linhas úteis.

DIVISION / SECTION	COUNT YES / NO	COMMENTS
IDENTIFICATION DIVISION	No	
ENVIRONMENT DIVISION		
CONFIGURATION SECTION	No	
INPUT-OUTPUT SECTION	No	
DATA DIVISION		
FILE SECTION	No	
WORKING STORAGE SECTION	No	
PROCEDURE DIVISION		
COMMENTS	No	
BLANK LINES	No	
SECTION NAMES (LABELS)	Yes	Count each section name as a LOC, including the line with the sentence PROCEDURE DIVISION.
EXECUTABLE COMMANDS	Yes	All executable commands shall be counted. If a command exceeds a single physical LOC, count each extra LOC as a separate LOC.
END COMMANDS (END-IF, END-PERFORM, EXIT etc.)	Yes	When a dot (.) appears alone in a command line, it shall not be counted.
INICIALIZATIONS	Yes	Initializations inside the Procedure Division, such as "MOVE ZEROS TO", shall be counted as LOC. If only one MOVE command is used to initialize several variables, count each physical line as a new LOC.
SQL STATEMENTS	Yes	Count each SQL statement exactly as any other COBOL command LOC.
EXTERNAL CALLS	Yes	The CALL statement shall be counted as any other COBOL command.

Tabela 13. COBOL LOC Counting *Standard.*

4.5 *PSPi0.2* - Size Categories Base

O elemento central do PSP, que permite ao desenvolvedor melhorar a sua capacidade de planejamento, é o método de estimativas PROBE. Conforme abordado no capítulo 3, ele se baseia no conceito de *proxy*, que nada mais é do que uma abstração que permite ao engenheiro de software obter o tamanho em LOC do programa que ele está planejando desenvolver.

Assim como o PROBE é o elemento central do PSP, as tabelas de categorias de tamanho relativo são os elementos centrais do PROBE. Através delas é que se consegue

obter o tamanho relativo em LOC/método de um objeto, conforme ilustram as tabelas propostas por Humphrey para C++ (Tabela 7, pág. 43) e para Object Pascal (Tabela 8, pág. 43). No entanto, quando o desenvolvedor inicia a utilização do PSP, além de ele normalmente não dispor de sua própria base de categorias, ele também não sabe como montá-la.

Por representarem um papel tão importante no roteiro das estimativas, sua montagem e validação deveriam fazer parte do processo padrão do PSP e não apenas serem sugeridas internamente no texto ou nos exercícios de programação. Optou-se, portanto, nesta dissertação, por criar um novo nível intermediário de processo, o *PSPi0.2*. Esse nível utiliza alguns elementos propostos no Anexo A de [HUMPHREY, 1995] e incorpora novos elementos, auxiliando o engenheiro de software, e, em última instância a própria empresa, a montar sua base de categorias inicial.

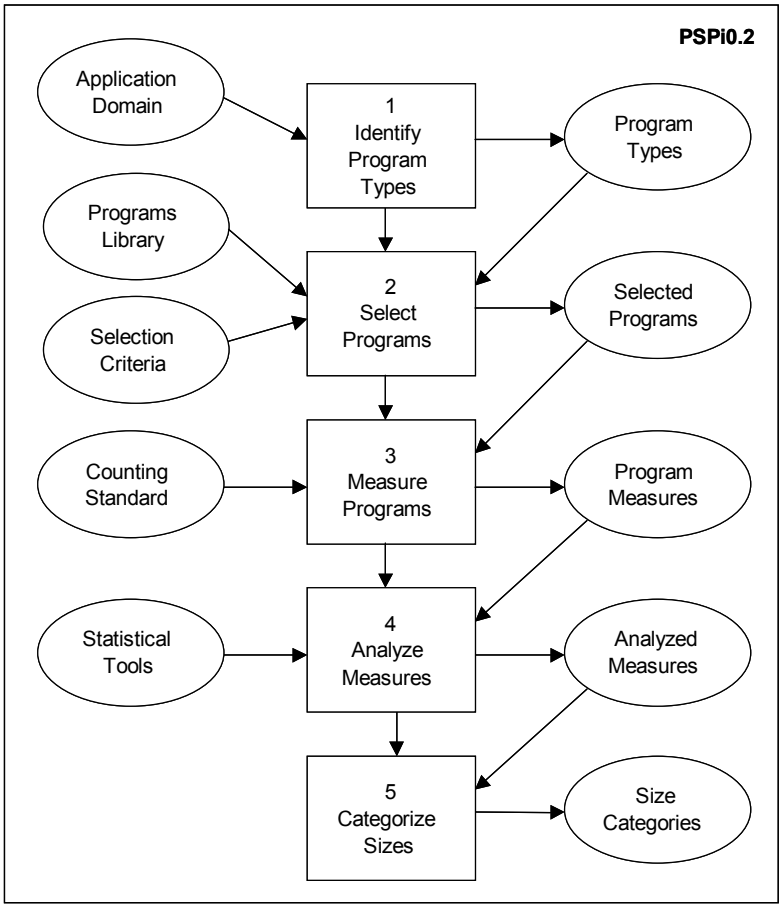

Figura 11. ***PSPi0.2 - Size Categories Base.***

A Figura 11 ilustra o processo *PSPi0.2* proposto e a Tabela 14 descreve cada um de seus passos. Conforme esclarecido no início deste capítulo, optou-se por introduzir os novos elementos em inglês, mantendo a compatibilidade com o PSP original.

Como o PROBE passa a ser utilizado a partir do nível PSP1 e a sua utilização pressupõe a existência das categorias de tamanho, optou-se por situar o nível intermediário *PSPi0.2* antes do nível PSP1.

Conforme se pode observar na Figura 11, o primeiro passo proposto é identificar, a partir do domínio da aplicação, os tipos de programas que farão parte da base de categorias. Cada domínio de aplicação possui suas características próprias, conforme já abordado anteriormente, o que diferencia a forma como os programas são desenvolvidos.

PHASE NUMBER	**PURPOSE**	To guide size categories base construction.
	ENTRY CRITERIA	• Programs library containing the programs developed using the same environment that the intended one.
1	IDENTIFY PROGRAM TYPES	• Identify program types according to the application domain. • Base this identification on program common characteristics.

2	SELECT PROGRAMS	• Select the programs to be measured that have the following characteristics:
		  • The application shall be one of the core business applications of the organization
		  • 90% of all batch programs shall be developed using the selected language
		  • Application shall be developed using the same practices actually being used by the organization (or, at least, as close as possible).

3	MEASURE PROGRAMS	• Count LOC as stated in COBOL LOC Counting *Standard*.
4	ANALYZE MEASURES	• Perform the graphical test to ensure data are normally distributed. • Perform the χ^2 test to confirm data are normally distributed
5	CATEGORIZE PROGRAMS	• Construct the size categories base based on procedures defined
	EXIT CRITERIA	• The size categories base tested and ready to be used by PROBE method.

Tabela 14. ***PSPi0.2 – Size Categories Base Process Script.***

Uma vez identificados os tipos de programas que caracterizam o domínio que se está tratando, deve-se selecionar os programas propriamente ditos. Para isso, toma-se como base a biblioteca de programas da organização, ou a da equipe de projeto, se for o caso, e os critérios de seleção estabelecidos, que encontram-se descritos na subseção 4.5.2.

Acompanhando o processo descrito na Tabela 14, o próximo passo é utilizar os padrões para a contagem de LOC e mensurar os programas que foram selecionados. Os padrões para a contagem na linguagem COBOL que são propostos por esta dissertação encontram-se descritos na Tabela 13. Com isso, o resultado é o conjunto de *proxies*, expresso na forma de LOC/*section* de cada programa.

Para que as categorias de tamanho possam ser montadas, os dados obtidos devem ser primeiramente avaliados utilizando as distribuições estatísticas apropriadas para, posteriormente, serem categorizados. Os procedimentos para efetuar essa análise encontram-se descritos na subseção 4.5.4 e os procedimentos para a montagem das categorias, na subseção 4.5.5.

O resultado que se obtém do processo *PSPi0.2*, conforme descrito na Tabela 14, é a tabela de categorias de tamanho relativo a ser utilizada nas estimativas de tamanho do método PROBE. Essas tabelas fornecerão o tamanho médio, em LOC/*section*, de cada uma das categorias de tamanho de *section*[35]: muito pequena (VS), pequena (S), média (M), grande (L) e muito grande (VL). Essas categorias ajudarão o desenvolvedor a estimar o tamanho do programa.

4.5.1 Tipos de programa do ambiente corporativo

Cada domínio de aplicação possui características próprias que acabam por se refletir nos tipos de programa que o compõem. Por exemplo, um sistema embutido, em geral, não necessita de uma entrada de dados on-line, mas, por outro lado, necessita de um algoritmo otimizado que lhe confira melhor performance. Um desenvolvedor desse ambiente não necessita saber quantas LOC são necessárias para desenvolver uma entrada de dados, porém, precisará estimar quantas LOC um algoritmo complexo deve conter.

[35] Optou-se por manter as siglas das categorias do original em inglês: VS (*very small*), S (*small*), M (*medium*), L (*large*) e VL (*very large*), para ser compatível com o restante do material original.

Ou seja, existem tipos de programas que são característicos de cada domínio e/ou ambiente e que não fazem sentido em outro. Por esse motivo, a primeira etapa na montagem da base de categorias é identificar os tipos básicos de programas que as irão compor. Conhecê-los, e à sua estrutura, facilita não apenas a tarefa de estimativa, como o desenvolvimento do programa em si.

No âmbito das aplicações comerciais que processam no ambiente *mainframe* pode-se considerar três tipos básicos de programa:

- Receptor: programa que recebe dados diretamente do usuário final, de forma on-line, com a finalidade ou não de armazenamento. No ambiente corporativo de grande porte, em geral, esses programas são desenvolvidos usando linguagens de quarta geração ou linguagens de terceira geração associadas a serviços de um monitor transacional.
- Atualizador: programa *batch* que, em geral, faz o processamento e atualização de grandes volumes de informação, envolvendo o negócio da aplicação propriamente dito, atualizando as bases da aplicação.
- Formatador: programa *batch* que faz a extração de informações das bases da aplicação e as disponibiliza para entidades externas, quer na forma de consulta on-line, relatório impresso, ou outro meio magnético qualquer (fitas, arquivos transmitidos etc.).

Não será foco deste trabalho tratar a porção on-line dos ambientes corporativos de grande porte, conforme abordado anteriormente. Portanto, os programas do tipo Receptor não serão considerados para compor a base de categorias de tamanho do PSPi. Serão, portanto, considerados apenas os programas dos tipos: Formatador e Atualizador.

4.5.2 Seleção do conjunto de programas

Para uma empresa que ainda não possua sua base de métricas consolidada, ou para o desenvolvedor individual que ainda não possua as suas próprias estimativas, é necessário selecionar um conjunto de programas existente para que possa compor uma base inicial de estimativas, até que a base própria esteja desenvolvida. Para isso, é preciso primeiramente estabelecer critérios mínimos para a seleção.

Considerando o universo que se pretende atender, ou seja, as empresas com sistemas legados em ambiente *mainframe*, foram estabelecidos os seguintes critérios:

- A aplicação deve pertencer a algum segmento da linha de frente de negócios da empresa, de modo a refletir suas regras de negócio e seu ambiente de operação;
- A parte *batch* da aplicação, composta por programas das categorias Formatador e Atualizador, deve possuir a maioria dos programas desenvolvidos em COBOL, garantindo que essas categorias estejam representadas no conjunto;
- A aplicação deve ter uma idade tal que as práticas utilizadas na época não sejam significativamente diferentes das atualmente em vigor na empresa.

Na verdade, esse constitui um conjunto mínimo de critérios para a seleção de um conjunto de programas inicial. O ideal é que todos os programas tivessem sido desenvolvidos pelo próprio desenvolvedor que utilizará a base. Porém, isso não constitui um problema, conforme se verá mais adiante nesse capítulo.

4.5.3 Mensuração dos programas selecionados

Após terem sido identificados os programas que farão parte da base de categorias de tamanho, seguindo os critérios definidos na seção anterior, e de tê-los classificado de acordo com os tipos básicos de programa do ambiente, é necessário então mensurá-los.

Para medir o tamanho do programa deverá ser aplicado o padrão para a contagem de LOC definido para o PSPi, COBOL LOC Counting *Standard*, apresentado na Tabela 13 da seção 4.4, de modo a obter as informações que comporão a base de *proxies*. As seguintes informações deverão ser obtidas:

- A quantidade total de LOC úteis da *Procedure Division*;
- A quantidade total de *sections* que compõem a *Procedure Division;* e
- A quantidade de LOC/*section.*

De forma análoga às tabelas originais de categorias de tamanho propostas por Humphrey, que utilizam como *proxy* a quantidade de LOC/método, as tabelas de categorias propostas no PSPi utilizarão a quantidade de LOC úteis da *Procedure Division*/*section*, ou simplesmente, LOC/*section.*

4.5.4 Análise da base de programas mensurados

A maioria dos procedimentos utilizados nas estimativas de tamanho de programa e gerenciamento da qualidade de software envolvendo métodos estatísticos, incluindo o próprio método PROBE, parte do princípio que os dados seguem a distribuição normal[36] [HUMPHREY, 1995]. Por isto, antes de utilizar um conjunto de dados com o propósito estatístico de estimativas, é necessário analisar o quanto a distribuição destes dados se aproxima da distribuição normal.

Essa verificação pode ser feita de duas formas:

- Forma gráfica: comparando visualmente o gráfico da distribuição dos dados que estão sendo analisados, com o gráfico da distribuição normal;
- Teste do χ^2: comparando os dados que estão sendo analisados, com os dados da distribuição χ^2 que pode ser utilizada, entre outras finalidades, para verificar se os dados provêm de uma distribuição em particular, no caso, da distribuição normal.

Caso a distribuição dos dados que estão sendo analisados não se aproxime da distribuição normal, será necessário executar transformações nesses dados, visando aproximar sua distribuição da distribuição normal, utilizando, por exemplo, o artifício de transformá-los usando seu logaritmo neperiano.

No contexto desta dissertação, para analisar se o conjunto de programas mensurados pode ser tratado como uma distribuição normal, para efeito de compor as categorias de tamanho dos *proxies*, é recomendado executar ambos os testes sobre o conjunto de LOC/*section* obtido.

Apesar de os procedimentos para executar os testes encontrarem-se descritos no Anexo A de [HUMPHREY, 1995], optou-se por incluí-los nesse material uma vez que são parte importante do processo *PSPi0.2*. No entanto, optou-se por lhes dar um novo formato, no qual as equações aparecem destacadas dos procedimentos e as explicações são mais aderentes aos dados do modelo. Também esses elementos aparecem em inglês, mantendo a compatibilidade com o PSP original.

[36] Para informações detalhadas sobre a distribuição normal (ou Gaussiana), bem como outros tipos de distribuição, consultar [SPIEGEL, 1977].

A Tabela 15, apresentada a seguir, traz o conjunto de fórmulas estatísticas que é utilizado para analisar a distribuição dos dados e que consiste, basicamente, nas medidas que caracterizam a distribuição:

- Média;
- Desvio padrão; e
- Variância.

Além disso, a tabela contempla as medidas que são usadas para apoiar os testes:

- Termo normal; e
- Fator Q (também conhecido como χ^2).

Essas fórmulas são referenciadas nos procedimentos para a execução do teste gráfico, apresentado na Tabela 16 e nos procedimentos para a execução do teste do χ^2, apresentado na Tabela 17.

NAME	FORMULAS	TERMS
AVERAGE	$x_{avg} = \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ **Formula 1.** ***Average Calculation.***	x_{avg} ou μ = average n = number of items x_i = items
VARIANCE	$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - x_{avg})^2$ **Formula 2.** ***Variance Calculation.***	σ^2 = variance x_{avg} = average n = number of items x_i = items
STANDARD DEVIATION	$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - x_{avg})^2}$ **Formula 3.** ***Standard Deviation Calculation.***	σ = standard deviation x_{avg} = average n = number of items x_i = items
NORMAL FORM	$z_i = \frac{(x_i - x_{avg})}{\sigma}$ **Formula 4.** ***Normal Form Transformation.***	z_i = normal item x_i = items x_{avg} = average σ = standard deviation
Q FACTOR	$Q = \sum_{i=1}^{S}\frac{(N_i - k_i)^2}{N_i}$ **Formula 5.** ***Q Factor Calculation.***	Q = factor to be used in χ^2 table S = number of segments i = segment N_i = number of normal items in the segment ki = number of dataset items in the segment

Tabela 15. ***PSPi0.2 - Statistical Formulas.***

GRAPHICAL TEST FOR NORMALITY PROCEDURES	
PURPOSE	• To guide the execution of the graphical test for normality, as a part of the size categories base construction.
ENTRY CRITERIA	• Programs measures, sorted in increasing order of LOC/section. • A table of the normal distribution (Appendix A of [HUMPHREY, 1995] or an automated statistical tool.
1	• Calculate the average of the LOC/section to be tested, using Formula 1.
2	• Calculate the variance of the LOC/section, using Formula 2. • Consider: *n* (for the whole population) ou *n-1* (for a sample of the whole population).
3	• Calculate the standard deviation of the LOC/section, using Formula 3. • The standard deviation can also be calculated by taking the square root of the variance.
4	• Obtain the normal form of all LOC/section items using Formula 4.

5	• Construct a LOC/section data table, containing the following columns:
	  • LOC/section in ascending order;
	  • Item number *i* (meaning the position that item occupies in the ascending ordered list)*;*
	  • Cumulative fraction calculated as *i/n* (where *n* is the total number of items)*;*
	  • Normal form of the item z_i.

6	• Construct a normal table, containing the following columns:
	  • *z* values from –4,0 to +4,0 in increments of 0,2;
	  • CDF of the normal distribution calculated using [HUMPHREY, 1995] Appendix A or an statistical tool.

7	• For each *z* value of the normal table, find the nearest lower value *zi* in the LOC/section table and enter its cumulative fraction in the normal table, besides normal distribution CDF values. If it is no lower value, enter the value zero (0).
8	• Plot the normal distribution curve from the normal distribution CDF values. • Plot the LOC/section curve from the LOC/section distribution CDF values.
9	• Visually examine the quality of the fit.
EXIT CRITERIA	• The normal distribution curve. • The LOC/section curve.

Tabela 16. Procedimentos para o teste gráfico(adaptado de [HUMPHREY, 1995]).

Ao seguir os procedimentos recomendados na Tabela 16, o que se obtém é um gráfico comparando a CDF da distribuição normal com a CDF da distribuição dos dados de LOC/*section* que serão utilizados para montar a base de categorias de tamanho. Embora não tenha a mesma precisão do teste do χ^2, o teste gráfico pode dar um primeiro sinal se será possível considerá-la como uma distribuição normal.

χ^2 TEST FOR NORMALITY PROCEDURES	
PURPOSE	• To guide the execution of the χ^2 test for normality, as a part of the size categories base construction.
ENTRY CRITERIA	• Programs measures, sorted in increasing order of LOC/section. • A table of the χ^2 distribution (Appendix A of [HUMPHREY, 1995] or an automated statistical tool.
1	• Calculate the average of the LOC/section to be tested, using Formula 1.
2	• Calculate the variance of the LOC/section, using Formula 2. • Consider: *n* (for the whole population) ou *n-1* (for a sample of the whole population).
3	• Calculate the standard deviation of the LOC/section, using Formula 3. • The standard deviation can also be calculated by taking the square root of the variance.
4	• Obtain the normal form of all LOC/section items using Formula 4.

5	• Construct a LOC/section data table, containing the following columns:
	  • LOC/section in ascending order;
	  • Item number *i* (meaning the position that item occupies in the ascending ordered list)*;*
	  • Cumulative fraction calculated as *i/n* (where *n* is the total number of items)*;*
	  • Normal form of the item z_i.

6	• Determine the maximum number of segments *S* so that, if possible:
	  • *n/S* is an integer equal to or greater than 5;
	  • $S > 3$; and
	  • $S^2 \geq n$.
	• If necessary, pick *S* so that *n/S* is not an integer.
	• When you have two or more choices, select the one where the values of *n* and S^2 are most nearly equal.

7	• Divide the probability range of the normal distribution into *S* equal segments. Do this by dividing the normal distribution into *S* equal segments that each have a probability of *1/S.*

8	• Construct a normal table, containing the following columns:
	  • Upper and lower bounds for each segment of the normal distribution;
	  • The expected number of items N_i, in each segment. For equal-sized segments, all the N_i equal *n/S*;
	  • The number of items k_i in the LOC/section table with *zi* values that fall within that segment's range are above the lower limit and less than or equal to the upper limit.
	  • Find by counting the items in the LOC/section table with the values z_i between the limits for that segment.

9	• Calculate the Q value using Formula 5.
10	• Look up the probability value *p* for this value of *Q* in the χ^2 table or calculate it using an automated statistical tool.
11	• Calculate the distribution tail as 1-p.
12	• If the tail area 1-p < 0.05: LOC/section certainly do not follow the normal distribution; • If the tail area 1-p > 0,2: LOC/section certainly follow the normal distribution. • If the tail area 1-p is between 0,05 and 0,2 it means intermediate degrees of fit.

Tabela 17. Procedimentos para o teste de χ^2 (adaptado de [HUMPHREY, 1995]).

Analogamente, ao aplicar os procedimentos descritos na Tabela 17, o que se faz é utilizar a técnica do teste do χ^2 para determinar, matematicamente, o quanto a distribuição dos dados de LOC/*section* se aproxima da distribuição normal.

Uma vez confirmado que se pode tratar os dados como uma distribuição normal, já é possível montar a base de categorias de tamanho relativo, cujos procedimentos serão descritos na próxima seção.

Pode ocorrer, no entanto, que os dados não sigam a distribuição normal e ser então necessário efetuar sobre eles uma transformação, de modo que possam ser utilizados. Nesse caso, o artifício sugerido por [HUMPHREY, 1995] é o de transformar as LOC/*section* utilizando o seu logaritmo neperiano, ou seja, aplicando: *ln (LOC/section)*. Em seguida, todos os cálculos devem ser refeitos e apenas no momento de obter efetivamente o tamanho médio de cada uma das categorias dos *proxies* é que se retorna o valor ao valor original LOC/*section*.

4.5.5 Montagem das categorias de tamanho dos proxies

Uma vez selecionados os programas, contadas as LOC/*section* e confirmado que podem ser tratados como uma distribuição normal, resta apenas montar as categorias de tamanho relativo.

Optou-se, nesta dissertação, por manter a mesma nomenclatura utilizada nas tabelas para C++ e Object Pascal descritas em [HUMPHREY, 1995] e apresentadas no capítulo 3, Tabela 7 e Tabela 8, respectivamente:

- VS (*Very Small*) = Muito pequena
- S (*Small*) = Pequena
- M (*Medium*) = Média
- L (*Large*) = Large
- VL (*Very Large*) = Muito Grande

A Tabela 18 demonstra como são calculados os valores médios de cada uma dessas categorias. Como se pode observar, esses valores são provenientes de cálculos efetuados em torno da média da distribuição.

O valor da categoria M (*medium)* corresponde à média da distribuição e os valores das categorias S *(small)* e VS *(very small)* correspondem, respectivamente, à média menos um desvio padrão e à média menos dois desvios padrões, assim como os

valores das categorias L *(large)* e VL *(very large)* correspondem, respectivamente, à média mais um desvio padrão e à média mais dois desvios padrões.

O cálculo desses valores segue as orientações de [HUMPHREY, 1995].

CATEGORIA	SIGLA	VALOR MÉDIO LOC/*SECTION*
Muito pequeno	VS	$\mu - 2 * \sigma$
Pequeno	S	$\mu - \sigma$
Médio	M	μ
Grande	L	$\mu + \sigma$
Muito Grande	VL	$\mu + 2 * \sigma$

Tabela 18. Cálculo do valor médio das categorias de tamanho.

O que se propõe nesta dissertação é que se monte uma tabela de categorias de tamanho para cada categoria de programa do ambiente legado corporativo, identificada na seção 4.5.1. Assim, ter-se-á:

- Uma tabela para a categoria de programas do tipo Formatador, baseada em programas do tipo Formatador; e
- Uma tabela para a categoria de programas do tipo Atualizador, baseada em programas do tipo Atualizador.

4.6 *C36i - PROBEi Size Estimating Guidance*

Ao iniciar a utilização do método PROBE do PSP original, e conseqüentemente o preenchimento do formulário *C39 – Size Estimating Template*[37], a primeira dificuldade que o engenheiro de software enfrenta é a elaboração conceitual do programa e, em seguida, a seleção da categoria de tamanho relativo mais adequada para cada *proxy*. É possível que, mesmo tendo experiência em desenvolvimento no ambiente em questão, o desenvolvedor não tenha experiência em estimar tamanho de programa ou até mesmo em produzir previamente um projeto conceitual.

Como o método PSP original não provê qualquer tipo de ajuda adicional ao desenvolvedor nesse sentido, optou-se, na instância PSPi, por desenvolver um conjunto de orientações, agrupadas na forma de um guia. Esse guia, denominado *C36i – PROBEi*

[37] *C39 – Size Estimating Template,* contido no Anexo C de [HUMPHREY, 1995] e já mencionado no capítulo anterior dessa dissertação.

Size Estimating Guidance[38], apresentado na Tabela 19, visa auxiliar a identificação das *sections* que comporão o programa, bem como a seleção do tamanho relativo típico de cada *section*, de acordo com o tipo de programa que se está desenvolvendo. Dessa forma, se pretende auxiliar o desenvolvedor, tanto no projeto conceitual do programa, quanto na estimativa de seu tamanho.

A elaboração do guia *C36 – PROBEi Size Estimating Guidance* envolveu duas atividades distintas:

- Identificação das *sections* típicas; e
- Identificação do tamanho típico das *sections.*

A primeira atividade, identificação das *sections* típicas, envolveu a descoberta das estruturas básicas recorrentes, presentes nos programas COBOL das aplicações comerciais de um ambiente *batch* de grande porte. Essas estruturas foram mais facilmente visualizadas ao se analisar separadamente os programas do tipo Formatador e Atualizador. Dessa observação surgiram as duas primeiras colunas da Tabela 19, que expressam o tipo de *section* e sua utilização.

A segunda atividade, identificação do tamanho típico das *sections,* envolveu a descoberta de como se comportam, em termos de tamanho, as *sections* de programas COBOL do ambiente legado corporativo. Para isso, foi necessário utilizar o resultado da primeira etapa do estudo de caso[39]: a base de categorias de tamanho relativo dos *proxies*. Note-se que, conforme já havia sido citado no início deste capítulo, como as alterações propostas por esta dissertação encontram-se no terceiro nível de abstração do PSP, Instanciação para Projetos, muitas vezes o estudo de caso e o próprio modelo se intercalam, como no caso dessas recomendações para o tamanho típico das *sections.*

[38] Lembrando que se optou por inserir os novos elementos em inglês para manter a compatibilidade com os demais elementos do PSP original.
[39] O estudo de caso é descrito em detalhes no Capítulo 6 desta dissertação.

SECTION	DESCRIPTION	SUGGESTED RELATIVE SIZE
Main section	• A section that usually calls the other ones and holds the command GOBACK. • Consider one main section for each program.	S
Starting section	• A section that has a standard set of parameter initializations such as: date, hour etc. • Consider one starting section for each program. • Its size may vary according to the amount of parameter initializations needed.	M
Normal ending section	• A section that has commands used when a normal processing had occurred, such as displaying a standard message area. • Consider one normal ending section for each program.	S
Abnormal ending section	• A section that has commands used when an error had occurred during the processing, such as: the move commands used to initialize a standard message area and the related display. • Consider one abnormal ending section for each program.	S
Open section (Any kind of file or database table)	• A section that has the OPEN command and its related return test. • Consider one open section for each input file, input database table, output file and output database table.	S
Close section (Any kind of file or database table)	• A section that has the CLOSE command and its related return test. • Consider one close section for each input file, input database table, output file and output database table.	S
Read section (sequential files and VSAM files)	• A read section of a sequential file or a VSAM file has the READ command and its related return test. • Consider at least one read section for each input file.	S
Read section (database tables)	• A read section of a database table has the SELECT command and its related return test. • Consider at least one read section for each read only database table.	M
Write section (sequential files, VSAM files and report files)	• A write section of a sequential file, a VSAM file or a report file has the WRITE command and its related return test. • Consider at least one write section for each output file and for each input-output file. • Additional write sections are needed if the file has more than one type of record (such as: header record, standard record, trailer record). Consider at least one write section for each type of record.	S

SECTION	DESCRIPTION	SUGGESTED RELATIVE SIZE
Write section (database tables)	• A write section of a database table has the UPDATE or INSERT commands and their related return test. • Consider at least one write section for each updateable database table.	M
External call section (subroutines)	• A call section includes the input parameters settings, the CALL command itself and its related return test. • Consider one external call section for each different subroutine. • The size may vary according to the amount of entry parameters to be set.	S to M
Report control section	• A section that has commands needed to properly control report paging (such as line and page counters, header and footer printing etc.)	S
Selection section	• A section that has the procedures needed to properly ignore unnecessary records, according to business rules or to ensure that data meet such business rules. • One S section usually holds 3 selection conditions, considering one IF command for each selection criteria.	3 / S
Calculation section	• A section that has the procedures needed to transform data, according to business rules. • One S section usually holds 5 calculation commands, considering one COMPUTE or equivalent command for each calculation need.	5 / S
Update Logic section	• A section that holds the logical commands needed in order to properly address a file or table update. • Although an update logic section may appear in a Formatter program it is more likely to appear in an Updater one. • One S section usually holds 15 MOVE or equivalent commands.	15 / S

Classification section (using internal SORT)	• Classification using internal SORT has two sections:
	  ◦ The input sort section, which may vary from VS to VL, according to the processing needs.
	  ◦ The output sort section, which also may vary from VS to VL, according to the processing needs.
	• Generally, the complexity is located in one of them (if the input section is VL, the output one is VS and if the input section is VS, the output one is VL).
		VS to VL

Tabela 19. ***C36i - PROBEi Size Estimating Guidance.***

O PSP parte do princípio que os requisitos são conhecidos, isso quer dizer que ao iniciar o desenvolvimento de um programa, se conhece: as informações de entrada e saída, as informações que o programa precisará acessar de outras aplicações e as regras de negócio. A partir dessas informações, e utilizando as orientações propostas na Tabela 19, é possível estimar o tipo e o tamanho relativo das *sections* que serão necessárias. Essas orientações consistem, basicamente em:

- Todo programa, independentemente do seu tipo, possui, no mínimo: uma *section* principal que chama as demais; uma *section* de inicialização de parâmetros como data, hora, área de mensagens etc.; uma *section* para tratamento do término normal do programa que inclui a exibição de uma área padrão de mensagens e o fechamento dos arquivos; e uma *section* para término anormal de programa, contendo a carga e exibição das mensagens de erro. Todas elas podem ser consideradas inicialmente de tamanho pequeno (S), podendo variar de acordo com o estilo de programação adotado.
- Arquivos de entrada ou de saída do tipo VSAM ou seqüencial: para esses arquivos deverá ser previsto, no mínimo, uma *section* para abertura e uma para fechamento. Se o arquivo for de entrada, uma *section* adicional para a leitura dos registros e se for de saída, uma para gravação. Essas *sections* incluem o comando em si e o respectivo teste de retorno (também conhecido como teste de *status*) e serão de tamanho pequeno (S). O tratamento dos dados em si deve ser previsto em outras *sections*, como por exemplo, *sections* de cálculo, seleção etc.
- Relatórios impressos: encaixam-se na categoria de arquivos de saída seqüenciais e para eles, além das *sections* descritas no item anterior, também se deve prever uma *section* de controle (das páginas, quantidade de linhas, cabeçalho, rodapé etc.) de tamanho pequeno (S).
- Arquivos de entrada ou de saída do tipo tabela de banco de dados: assim como os tipos anteriores de arquivo, as tabelas de banco de dados também necessitam de uma *section* para abertura, outra para fechamento e outra para leitura ou gravação, conforme o caso. Porém, por necessitarem incorporar estruturas adicionais da linguagem SQL para o acesso ao banco de dados, as *sections* de leitura e de gravação possuem, no mínimo, tamanho médio (M).

Da mesma forma que no caso anterior, essas *sections* incluem o comando em si e o respectivo teste de retorno.

- Acesso a informações de outras aplicações (via sub-rotina): em ambientes que trabalham de forma estruturada e organizada, o acesso a informações pertencentes a outras aplicações se dá por meio de chamadas a sub-rotinas, gerando uma espécie de encapsulamento dos domínios. Essas chamadas envolvem uma *section* que efetua a carga dos parâmetros de entrada, a chamada à sub-rotina propriamente dita e o tratamento do código de retorno correspondente. A categoria de tamanho poderá variar entre pequena (S) e média (M), dependendo da quantidade de parâmetros de entrada a serem fornecidos.
- Acesso direto a informações de outras aplicações: No caso do ambiente trabalhar de forma menos organizada, permitindo que se faça acesso direto a informações de outras aplicações, cada arquivo (tabela) acessado poderá ser considerado como um arquivo (tabela) da própria aplicação, devendo-se prever as *sections* correspondentes, conforme descrito nos itens anteriores.
- Lógica do negócio: a lógica do negócio pode envolver diferentes estruturas de programação, dificultando estabelecer orientações precisas a respeito de seu tamanho relativo. Apesar disso, algumas sugestões são fornecidas também na Tabela 19, levando-se em conta a necessidade de seleção (desprezar dados desnecessários no contexto do programa ou checar dados contra regras específicas de negócio), cálculos (transformação nos dados de entrada), lógica de atualização (tratamento dos dados para gravação) e classificação (usando o comando SORT interno).

As orientações aqui propostas visam auxiliar o engenheiro de software na sua tarefa de começar a estimar tamanho de programa COBOL utilizando o método PROBE. De forma alguma essas orientações poderão substituir a experiência do desenvolvedor na hora de prever os elementos que comporão o programa ou o seu tamanho, porém podem servir de ponto de partida na hora de conceber a solução e a estimativa. Com o tempo e a utilização freqüente, o próprio desenvolvedor poderá alterar as orientações, ajustando-as ao seu estilo de programação, aos padrões vigentes em sua organização ou a necessidades específicas de outros domínios de aplicação.

4.7 *C39i – Size Estimating Template*

No PSP original, conforme já havia sido abordado no capítulo 3 desta dissertação, o formulário que apoia a execução do método de estimativas PROBE é o *C39 - Size Estimating Template*, parcialmente ilustrado na Figura 12. Esse formulário, no entanto, não oferece orientações suficientes para que o desenvolvedor consiga selecionar o tamanho mais adequado para a *section*, nem tão pouco para que ele consiga definir quais tipos de *section* deverão ser utilizados[40].

Essa orientação adicional que o engenheiro de software necessita foi suprida, na instância PSPi, pelas orientações que esta dissertação propôs na seção anterior, através do guia *C36i - PROBEi Size Estimating Guidance*. No entanto, permaneceu um descompasso entre essas novas orientações e o registro das informações. Por isso, foi necessário alterar também o formulário que registra o planejamento de tamanho de programa, gerando uma nova versão, denominada *C39i – Size Estimating Template*, cujas áreas alteradas se encontram ilustradas na Figura 13.

Note que se optou, tanto na Figura 12 como na Figura 13, por representar apenas as áreas alteradas. O restante do formulário, que contempla os cálculos utilizando a regressão linear, podem ser consultados em [HUMPHREY, 1995].

BASE ADDITIONS	**TYPE**	**METHODS**	**REL SIZE**	**LOC**	**LOC**
TOTAL BASE ADD	**(BA)**	**=> => =>**	**=> => =>**		
NEW OBJECTS	**TYPE**	**METHODS**	**REL SIZE**	**LOC**	**LOC**
TOTAL NEW OBJ	**(NO)**	**=> => =>**	**=> => =>**		

Figura 12. *C39 - Size Estimating Template* – áreas que foram substituídas.

[40] O método original refere-se a objetos e tamanho relativo de objetos (LOC/método). Porém, no contexto desta dissertação esse conceito equivale a *sections* e tamanho relativo de *sections (LOC/section).*

No formulário original existem duas áreas onde são registradas as informações sobre a estimativa das LOC, conforme ilustra a Figura 12:

- *Base Additions (BA)*: área do formulário onde o desenvolvedor registra as estimativas referentes às LOC que pretende adicionar a um objeto já existente[41].
- *New Objects (NO)*: área do formulário onde o desenvolvedor registra as estimativas referentes às LOC de novos objetos que serão desenvolvidos.

No novo formulário *C39i – Size Estimating Template*, essas duas áreas foram integralmente substituídas pelas áreas propostas na Figura 13.

Optou-se por não alterar as duas abreviaturas utilizadas, BA e NO, porque são utilizadas em diversos outros pontos do método PSP original e seria necessário alterar uma série de formulários do método original apenas para alterar a nomenclatura. Portanto, ao utilizar a instância PSPi, o desenvolvedor deverá interpretar essas abreviaturas da seguinte forma:

- *Base Additions (BA)*: LOC que se pretende adicionar a uma *section* já existente no programa base, caso o desenvolvimento que está sendo estimado refira-se a alteração de um programa existente.
- *New Objects (NO)*: LOC que se pretende adicionar na forma de novas *sections*, seja a um programa existente, seja para o desenvolvimento de um novo programa.

A utilização conjunta do formulário alterado *C39i – Size Estimating Template* com as orientações propostas no guia *C36i – PROBEi Size Estimating Guidance*, facilita o trabalho de estimativas, bastando que o desenvolvedor indique, nos espaços em branco, quantas *sections* daquele tamanho relativo ele pretende utilizar. Ao final, soma a quantidade total de *sections* de cada tipo e multiplica pela quantidade de LOC/*section* sugerida pela tabela da base de categorias de tamanho relativo, de acordo com o tipo de programa que está sendo desenvolvido.

Note que no PSP original não havia qualquer orientação sobre como descobrir essas *sections*, já na instância PSPi essa orientação encontra-se no guia e no próprio formato novo do formulário, oferecendo uma grande contribuição ao desenvolvedor.

---

[41] Para detalhes sobre *Base Additions (BA)* e *New Objects (NO),* consultar o capítulo 3 desta dissertação ou [HUMPHREY, 1995].

BASE ADDITIONS	**VS**	**S**	**M**	**L**	**VL**
MAIN SECTION (S)					
STARTING SECTION (M)					
NORMAL ENDING SECTION (S)					
ABNORMAL ENDING SECTION (S)					
OPEN SECTION (S)					
CLOSE SECTION (S)					
READ SECTION - VSAM/SEQ (S)					
READ SECTION – DB (M)					
WRITE SECTION – VSAM/SEQ (S)					
WRITE SECTION – DB (M)					
EXTERNAL CALL SECTION (S to M)					
REPORT CONTROL SECTION (S)					
SELECTION SECTION (3/S)					
CALCULATION SECTION (5/S)					
UPDATE SECTION (15/S)					
CLASSIFICATION SECTION (VS to VL)					
OTHER SECTIONS (VS to VL)					
NUMBER OF SECTIONS					
LOC/SECTION					
TOTAL LOC/SECTION					
TOTAL BASE ADDITIONS (BA)	=>	=>	=>	=>	

NEW SECTIONS	**VS**	**S**	**M**	**L**	**VL**
MAIN SECTION (S)					
STARTING SECTION (M)					
NORMAL ENDING SECTION (S)					
ABNORMAL ENDING SECTION (S)					
OPEN SECTION (S)					
CLOSE SECTION (S)					
READ SECTION - VSAM/SEQ (S)					
READ SECTION – DB (M)					
WRITE SECTION – VSAM/SEQ (S)					
WRITE SECTION – DB (M)					
EXTERNAL CALL SECTION (S to M)					
REPORT CONTROL SECTION (S)					
SELECTION SECTION (3/S)					
CALCULATION SECTION (5/S)					
UPDATE SECTION (15/S)					
CLASSIFICATION SECTION (VS to VL)					
OTHER SECTIONS (VS to VL)					
NUMBER OF SECTIONS					
LOC/SECTION					
TOTAL LOC/SECTION					
TOTAL NEW SECTIONS (NO)	=>	=>	=>	=>	

Figura 13. ***C39i - Size Estimating Template*** **– substituições.**

4.8 *C58i - COBOL Code Review Checklist*

Um bom *checklist* para a revisão de programas é aquele que auxilia o desenvolvedor a encontrar, não apenas o maior número de erros presentes no código, como também o maior número de potenciais problemas de processamento, no menor espaço possível de tempo. Para isso, é importante que, não apenas o *checklist* seja aderente ao ambiente que está sendo tratado, como também, que seja aderente às próprias práticas de programação utilizadas.

Embora alguns elementos do *checklist* de revisão de código sejam genéricos, ou seja, independam da linguagem que se está revisando, a maioria é bastante aderente ao ambiente em questão, conforme demonstrado pelos trabalhos apresentados no capítulo 3, a saber [ALBUQUERQUE & MEIRA, 1997] e [SILVA & MASIERO, 1998], que alteram as propostas de *chekclist* do PSP original para instanciação para os seus ambientes Java e Powerbuilder, respectivamente.

No PSP original, conforme descrito no capítulo 3, existem dois *checklists* distintos para executar a revisão do programa:

- *C57 – C++ Design Review Checklist* que se destina a revisar o projeto físico do programa[42].
- *C58 – C++ Code Review Checklist* que se destina, como o próprio nome diz, a revisar o código, propriamente dito.

No entanto, nem sempre é fácil distinguir, no momento da revisão, se determinado problema é proveniente da codificação, ou do projeto do programa. Exemplo disso é um dos itens do *checklist* original da revisão de projeto, que recomenda: “todos os nomes especiais e tipos são clara e especificamente definidos”. Essa poderia ser interpretada como uma questão de linguagem, simplesmente, e não de projeto do programa. Por esse motivo, na instância PSPi optou-se por incorporar ao *checklist* de revisão de código, algumas orientações em relação à checagem da estrutura do programa.

[42] Esse formulário é substituído no nível PSP2.1 pelo, *C65 – C++ PSP2.1 Design Review Checklist*, que engloba os testes referente à máquina de estados.

SECTION	SUGGESTED	OK
DOCUMENTATION	• Verify that the header is properly inserted and updated in the IDENTIFICATION DIVISION as stated in *C29i - COBOL Coding Standard.* • Verify that all major sections have an adequate comment inserted before any other command. • Verify that all nested commands have the proper indentation.	
WORKING STORAGE SECTION	• Verify that all variables are properly declared in the WORKING STORAGE SECTION. • Verify that all variables names follow the recommendations stated in *C29i - COBOL Coding Standard.*	

VSAM AND SEQUENTIAL FILES	• Verify that all VSAM and sequential files are properly declared in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION, including files to be used as printed reports.
	• Verify that all VSAM and sequential files are properly defined in the FILE SECTION of the DATA DIVISION, including files to be used as printed reports.
	• Verify that all VSAM and sequential files have the related COPY declared in the WORKING STORAGE SECTION of the DATA DIVISION.
	• Verify that all VSAM and sequential files have:
	  • An open section;
	  • A close section;
	  • A write section (for output files and files declared as I-O files);
	  • A read section (for input files and files declared as I-O files).
	• Verify that to each operation, the return status is properly tested and all possible errors are handled.	

DATABASE TABLES	• Verify that all needed dataviews to access database tables are properly declared in the WORKING STORAGE SECTION.
	• Verify that all needed SQL Cursors are properly declared in the WORKING STORAGE SECTION.
	• Verify that all database table have:
	  • An open cursor section (if a cursor is used);
	  • A close cursor section (if a cursor is used);
	  • A write section (for dataviews with update permission);
	  • A read section (for dataviews with read-only permissions).
	• Verify that to each operation, the SQL return status is properly tested and all possible errors are handled.	

EXTERNAL CALLS	• Verify that for each subroutine accessed by the program, the proper COPY is declared in the WORKING STORAGE SECTION. • Verify that each subroutine has a call section. • Verify that all input parameter are properly loaded before the subroutine is invoked. • Verify that to each subroutine call, the return status is properly tested and all possible errors are handled.	
REPORT CONTROLS	• Verify that all reports have their controls properly coded, such as: line counters and page counters.	

GENERAL	• Check every line of code for:
	  • Command syntax and
	  • Proper punctation.	

STANDARDS	• Ensure the code conforms to the coding standards proposed in *C29i - COBOL Coding Standard.*	

Tabela 20. ***C58i - COBOL Code Review Checklist.***

O formulário proposto para ser utilizado como guia no momento da revisão de código é o *C58i – COBOL Code Review Checklist*, apresentado na Tabela 20. Conforme se pode observar, ele foi desenvolvido de forma a tornar-se aderente aos padrões de codificação propostos na Tabela 12 - *C29 COBOL Coding Standard* e ao padrão de contagem de LOC proposto na Tabela 13 - COBOL LOC Counting *Standard.*

Ao conduzir a revisão de programa, o desenvolvedor, ou um de seus pares[43], assinala ao lado do item aquele que já foi checado. É possível que o desenvolvedor queira que o seu *checklist* seja customizado para atender a necessidades específicas do programa que está sendo desenvolvido. Nesse caso, poderia ser prevista uma linha para cada item a ser verificado, como por exemplo, uma linha de *checklist* específica para cada comando necessário à manipulação do arquivo A (open A, close A, read A). Esse tipo de abordagem pode ser interessante, especialmente se houver um grande número de arquivos e tabelas sendo manipulado ou, se a lógica de negócio for tão complexa que justifique montar um *checklist* mais detalhado para revisá-la. De outro modo, o *checklist* fornecido será suficiente para auxiliar o engenheiro de software a promover a revisão de código e projeto físico de seus programas COBOL.

4.9 Contribuições da proposta

Conforme discutido no capítulo anterior, o terceiro nível de abstração do PSP, Instanciação para Projetos, sofre a influência de três fatores: Paradigma de Desenvolvimento, Ambiente de Desenvolvimento e Linguagem de Programação. Isto significa que, se qualquer uma destas variáveis for alterada, adaptações no processo serão necessárias, de modo a refletir a realidade da aplicação.

Nesta dissertação, isto significa que as alterações que estão sendo propostas neste nível de abstração são aderentes ao ambiente legado corporativo de grande porte. Isto não quer dizer que o modelo proposto não possa ser utilizado em outras circunstâncias. Apenas significa que, de acordo com as características do ambiente, este nível terá que sofrer adequações.

[43] Na realidade, como o modelo PSP é individual, a revisão de código deveria ser conduzida pelo próprio desenvolvedor. No entanto, nada impede que se opte por uma revisão por pares, a ser executada por outro desenvolvedor.

Neste capítulo foi apresentada a proposta da instância PSPi para o ambiente corporativo, inserindo novos elementos e alterando elementos existentes do PSP original. Esta proposta é inovadora no sentido que prevê a montagem de uma base de categorias de tamanho relativo de *proxies* derivada de um conjunto de programas da organização e não apenas do próprio indivíduo, como seria o usual. Além disto, consolida as melhores práticas de codificação para a linguagem COBOL, sugerindo um padrão para codificação, um padrão para a contagem de LOC e um *checklist* para a revisão de código. Oferece, ainda, como adicional ao modelo PSP, e um diferencial em relação a qualquer outra abordagem anterior, um guia para auxiliar a concepção dos programas e a respectiva estimativa de tamanho, *C36i - PROBEi Size Estimating Guidance*.

Melhorar a capacidade de estimativa do desenvolvedor é o primeiro passo que se pode dar rumo à melhoria do planejamento de projeto e, em última instância, rumo a cronogramas mais realistas, que favorecem o cumprimento de prazos. A proposta de um guia, reduzindo o fator subjetivo da estimativa, aumenta ainda mais os resultados já positivos relatados com o uso do PSP e relacionados no capítulo 3.

4.10 Considerações sobre o capítulo

O presente capítulo dedicou-se a apresentar a proposta desta dissertação, a instância PSPi e os elementos que estão sendo introduzidos e/ou alterados.

As idéias aqui apresentadas serão exercitadas através do estudo de caso apresentado no capítulo 6, com a montagem da base de categorias de tamanho relativo das categorias Formatador e Atualizador e a correspondente validação dessas.

CAPÍTULO 5 - PSPPlus-i - A FERRAMENTA PROPOSTA

Conforme comprovado pelas experiências de [DISNEY & JOHNSON, 1998], o uso do PSP sem uma ferramenta automatizada, além de mais sujeito a erros, pode causar uma sobrecarga de tarefas ao desenvolvedor que o desencoraje de persistir na utilização do método. O próprio SEI reconhece essa limitação quando, ao liberar a primeira versão do TSP, disponibilizou também uma ferramenta de apoio [HUMPHREY, 1999].

Em [WENCESLAU et alli, 2001], a ferramenta PSP2000 é apresentada como uma alternativa para a automatização dos procedimentos de preenchimento dos formulários do PSP. No entanto, apesar de bastante útil e completa, a ferramenta implementa o PSP original, e não oferece mecanismos que facilitem sua adaptação a outras instâncias.

A ferramenta desenvolvida por esta dissertação, PSPPlus-i, visa apoiar a execução dos processos que foram alterados pela instância PSPi e não a automatização de todo o processo PSP. Por isso, ela pode ser usada em conjunto com a ferramenta PSP2000, bastando que se desenvolva uma ligação entre elas. Não se pode considerar que o PSPPlus-i seja uma ferramenta completa, o mais adequado é encará-la como um protótipo para apoiar a instância proposta PSPi.

5.1 Ambiente

A ferramenta PSPPlus-i foi construída utilizando o ambiente integrado de desenvolvimento Powerbuilder 5.0 e o banco de dados Sybase SQL Anywhere 5.0, ambos da Sybase.

A ferramenta oferece interface gráfica amigável e auto-explicativa, com telas bastante aderentes ao processo proposto por esta dissertação.

5.2 Visão geral da ferramenta PSPPlus-i

A ferramenta PSPPlus-i foi construída com a finalidade de auxiliar o engenheiro de software a utilizar os elementos propostos pela instância PSPi. Sendo assim, foi utilizada para apoiar a execução dos processos do estudo de caso.

Os módulos da ferramenta foram organizados em dois grupos de acordo com a sua finalidade: Ferramentas e Consultas. O primeiro grupo, Ferramentas, visa auxiliar o desenvolvedor em três das principais atividades propostas pela instância PSPi:

- Mensuração dos programas selecionados;
- Montagem das categorias de tamanho dos *proxies*;
- Estimativa de tamanho de programa.

O segundo grupo, Consultas, como o próprio nome diz, permite que o desenvolvedor consulte as informações armazenadas no banco de dados através dos módulos anteriores:

- Consulta de programas mensurados;
- Consulta base de categorias de tamanho;
- Consulta estimativa de tamanho de programa.

Todos os módulos podem ser acessados através do menu caractere, conforme ilustrado na Figura 14, ou pelos ícones individuais da barra de ferramentas, conforme ilustrado na Figura 15. As próximas subseções detalham cada um dos módulos principais da ferramenta e apresentam um exemplo de utilização real, com dados do estudo de caso.

Figura 14. PSPPlus-i - Menu principal.

Figura 15. PSPPlus-i - Barra de Ferramentas.

5.3 Módulos do grupo Ferramentas

5.3.1 Mensuração dos programas selecionados

O primeiro módulo do grupo Ferramentas, oferecido pela ferramenta PSPPlus-i, é o que implementa a mensuração do tamanho de um programa, baseada no padrão de contagem de LOC estabelecido, COBOL LOC Counting *Standard.* Segundo esse padrão, seria necessário mensurar apenas a quantidade de LOC úteis da *Procedure Division* e a quantidade de *sections* do programa. No entanto, conforme descrito no capítulo anterior, prevendo a continuidade do trabalho proposto por esta dissertação, outras informações também estão sendo capturadas no momento da contagem do programa: LOC úteis, em branco e de comentários de todas as grandes divisões do programa; quantidade de arquivos; quantidade de *dataviews* e quantidade de *copys*.

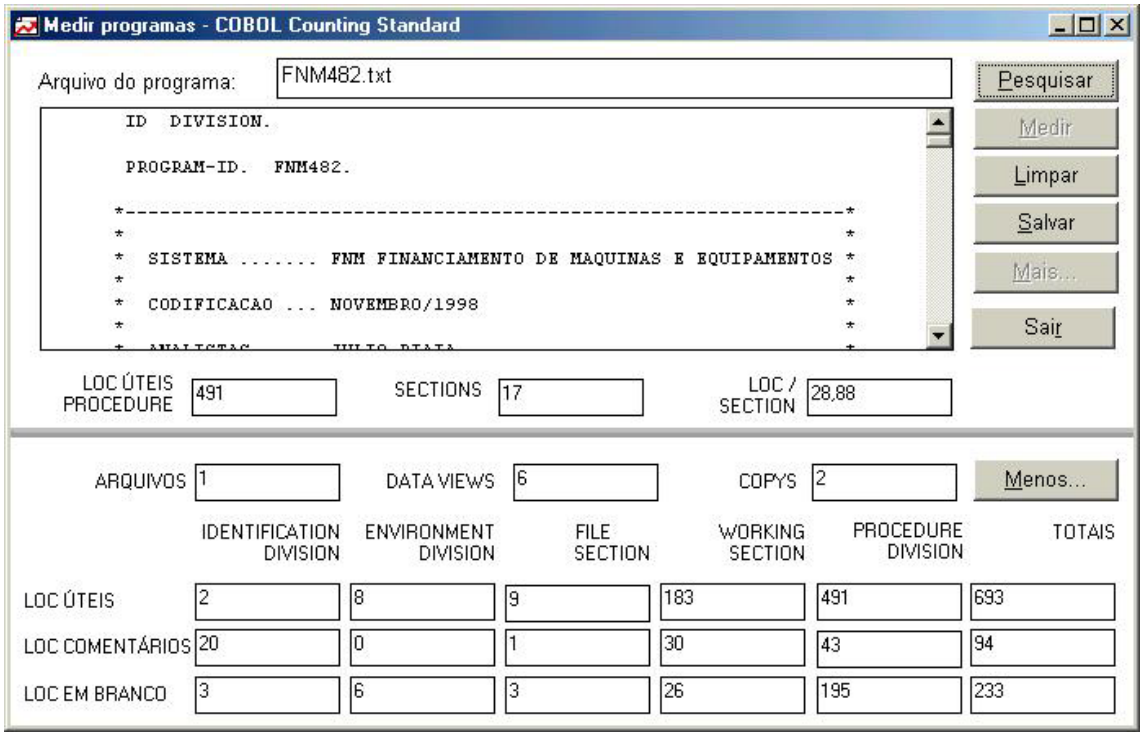

Figura 16. PSPPlus-i - Tela para mensuração de programas.

À medida que o programa vai sendo mensurado, as linhas de código vão sendo exibidas na tela para que o usuário possa analisá-las. Se houver interesse em incorporar o programa à base de programas, o usuário tem a opção de salvá-lo na base de dados.

Esse módulo da ferramenta pode ser utilizado, tanto no momento de montar a base de estimativas[44], como no momento de mensurar um programa com o objetivo de compará-lo com sua estimativa. No estudo de caso, esse módulo foi utilizado com as duas finalidades:

- Mensurar os 100 programas que originaram a categoria Formatador e os 30 programas que originaram a categoria Atualizador;
- Mensurar os 25 programas utilizados na validação da instância proposta.

A Figura 16 ilustra a mensuração efetuada para um dos programas que compõem a base de categorias de tamanho para a categoria de programas Formatador. Como se pode observar, a segunda parte da tela apresenta as informações adicionais que estão sendo armazenadas com o objetivo de aprofundar as análises em estudos futuros.

5.3.2 Montagem das categorias de tamanho dos *proxies*

O segundo módulo do grupo Ferramentas, disponibilizado pela ferramenta PSPPlus-i, tem a finalidade de auxiliar a montagem da base de categorias de tamanho[45]. Para isso, a ferramenta parte do pressuposto que os programas mensurados e catalogados na base já foram testados quanto à sua aderência à distribuição normal[46].

A ferramenta calcula o tamanho das categorias de tamanho: muito pequeno (VS), pequeno (S), médio (M), grande (L) e muito grane (VL). Para isso, utiliza a média (μ) e o desvio padrão (σ) dos programas mensurados e catalogados através do módulo apresentado na seção anterior.

Para o cálculo das categorias, duas opções são disponibilizadas:

- Cálculo normal: Calcula as categorias com base nas LOC/*section* dos programas que se encontram catalogados na base de dados;
- Cálculo com log neperiano: Calcula as categorias com base no logaritmo neperiano das LOC/*section* dos programas que se encontram catalogados na base de dados e depois procede a transformação inversa (para o caso

[44] Essa é a terceira atividade do processo proposto PSPi0.2 e é descrita na subseção 4.5.3 do capítulo 4.
[45] Essa é a quinta atividade do processo proposto PSPi0.2 e é descrita na subseção 4.5.5 do capítulo 4.
[46] Os testes, descritos em detalhes no capítulo 4, podem ser realizados por ferramentas que implementem análises estatísticas, bastando que se exporte a base de programas mensurados.

de os dados não seguirem exatamente a distribuição normal ou o cálculo apresentar valores negativos).

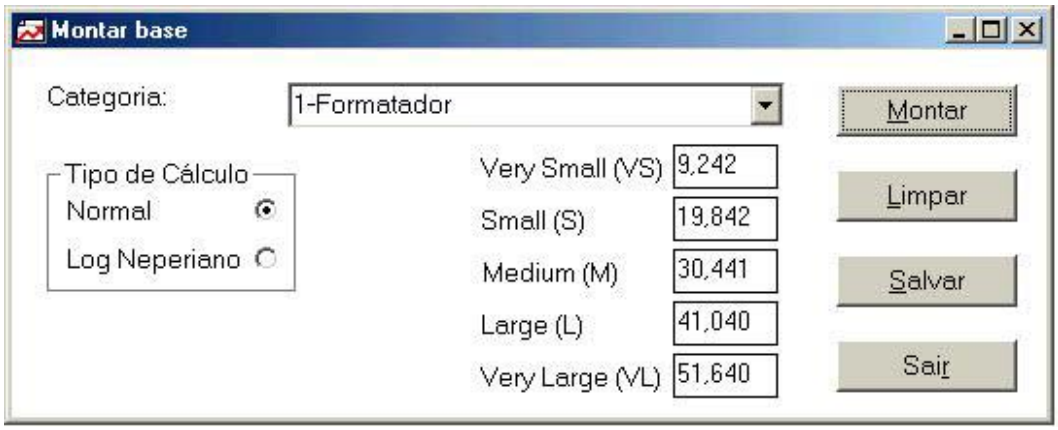

Figura 17. PSPPlus-i - Montagem das categorias de tamanho.

A Figura 17 ilustra a tela que implementa essa funcionalidade. Como se pode observar, o usuário tem a opção de fazer os cálculos, tanto utilizando a modalidade cálculo normal, quanto a modalidade cálculo com log neperiano; e, depois, salvar aquela que considere como mais adequada.

Essa figura ilustra a montagem da base de categorias para os programas do tipo Formatador, utilizando os dados reais do estudo de caso, conforme apresentado no capítulo 6.

5.3.3 Estimativa de tamanho de programa

O terceiro módulo do grupo Ferramentas, oferecido pela ferramenta PSPPlus-i, auxilia o desenvolvedor na tarefa de estimar o tamanho do programa utilizando as orientações que foram propostas na instância PSPi.

A Figura 18 ilustra a estimativa efetuada para um dos programas que foi utilizado no estudo de caso como validação da base de categorias de tamanho. Os dados desse mesmo programa aparecem preenchidos no formulário *C39i – Size Estimating Template* (que aparece no próximo capítulo, na Tabela 33).

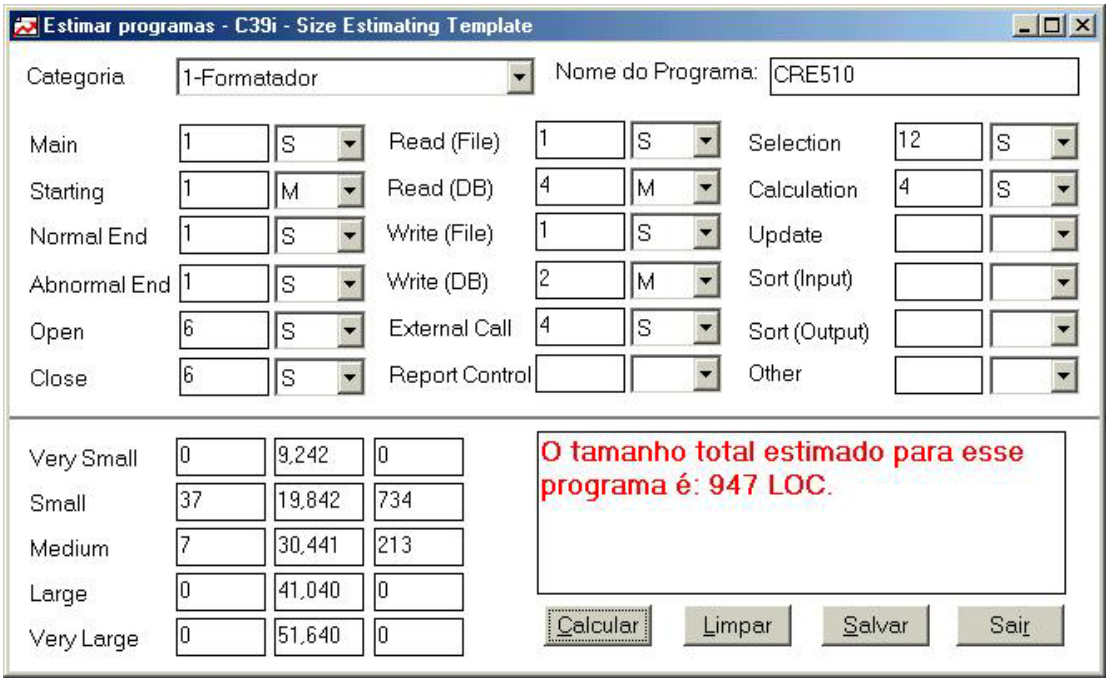

Figura 18. PSPPlus-i - Tela para estimativa de tamanho de programa.

Da mesma forma que os módulos anteriores, esse módulo também oferece ao usuário a possibilidade de registrar sua estimativa na base de dados, para posterior comparação com o tamanho real do programa.

Os valores de LOC/*section* são obtidos na base de categorias de tamanho, de acordo com o tipo de programa que o usuário selecionar: Formatador ou Atualizador.

5.4 Módulos do grupo Consultas

A ferramenta PSPPlus-i oferece ao desenvolvedor três tipos de consulta: aos programas mensurados, à base de estimativas e às estimativas efetuadas. Caso o usuário necessite, é possível exportar as informações que estão armazenadas nas bases relacionais do SQL Anywhere e manipulá-las através de outras ferramentas, como planilhas e editores de texto. Dessa forma, existe a flexibilidade de formatá-las de acordo com a necessidade, gerando, inclusive gráficos e planilhas.

5.4.1 Consulta programas mensurados

O primeiro módulo do grupo Consultas oferece ao usuário a possibilidade de consultar os dados de programas que tenham sido mensurados utilizando a opção de mensuração do grupo de Ferramentas.

A Figura 19 apresenta dados de programas que foram utilizados no estudo de caso para compor a categoria de tamanho dos *proxies* dos programas do tipo Atualizador.

A consulta oferece, além do nome do programa e sua descrição, também a quantidade de LOC úteis da *Procedure Division*, a quantidade de *sections* e a quantidade de LOC/*section.*

Figura 19. PSPPlus-i - Tela para consulta de programas mensurados.

5.4.2 Consulta categorias de tamanho

O segundo módulo do grupo Consultas oferece a consulta do tamanho de cada uma das categorias que compõem a base de tamanho de *proxies*.

A Figura 20 apresenta os dados para as categorias de programa do tipo Formatador e Atualizador, que foram obtidos utilizando o módulo de montagem da base do grupo de Ferramentas. Esses valores são provenientes do estudo de caso que é detalhado no próximo capítulo.

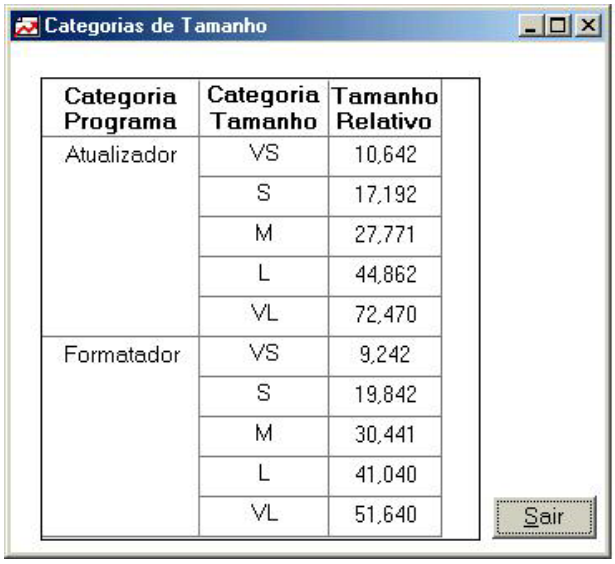

Figura 20. PSPPlus-i - Tela para consulta das categorias de tamanho dos *proxies*.

5.4.3 Consulta estimativa de tamanho de programa

O terceiro módulo do grupo Consultas oferece ao desenvolvedor uma forma de consultar as estimativas que ele tenha realizado através do módulo de estimativas do grupo Ferramentas do PSPPlus-i.

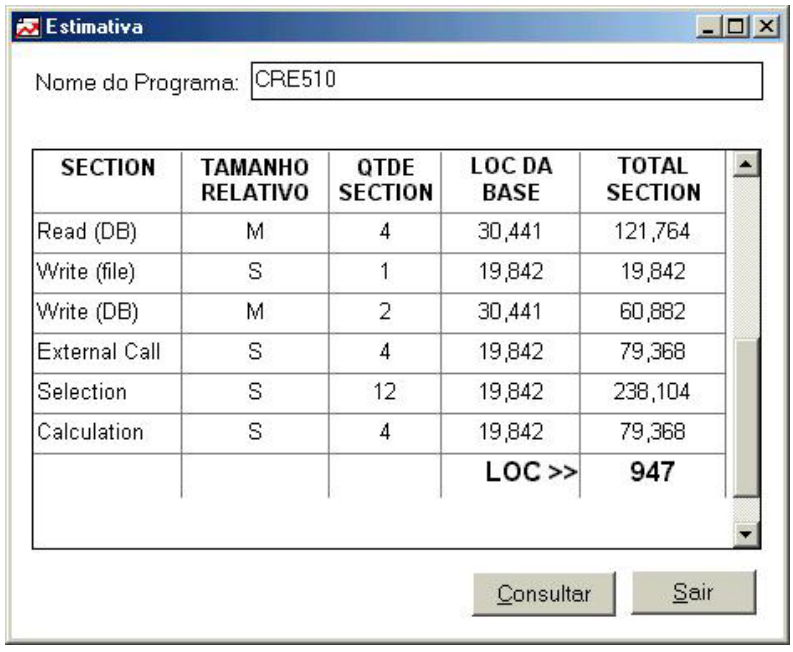

Figura 21. PSPPlus-i - Tela para consulta de estimativa de tamanho de programa.

A Figura 21 ilustra a consulta aos dados de estimativas de um dos programas que foram utilizados na validação da base de categorias de tamanho e que foi estimado utilizando o módulo de estimativas do grupo de Ferramentas (Figura 18).

5.5 Benefícios

O uso da ferramenta gráfica PSPPlus-i facilita a montagem da base de categorias de tamanho relativo dos *proxies*, no caso LOC/*section*, através dos módulos de mensuração de programas e de montagem da base, propriamente dita. Além disso, auxilia o processo de estimativas de tamanho, através da disponibilização das partes alteradas do formulário *C39i – Size Estimating Template* de forma on-line.

A ferramenta PSPPlus-i oferece ao engenheiro de software um apoio indispensável na utilização dos processos propostos por esta dissertação, permitindo:

- Redução na quantidade de erros que o processo manual está sujeito;
- Economia de tempo pela agilização das tarefas;
- Flexibilidade de manipulação das informações que podem ser facilmente exportadas do banco de dados SQL Anywhere para qualquer outra ferramenta (como planilhas eletrônicas etc.).

5.6 Oportunidades de melhoria

A ferramenta PSPPlus-i oferece algumas oportunidades de melhoria, conforme será discutido no capítulo 7, seção de trabalhos futuros:

- Desenvolvimento de mais um módulo, no grupo Ferramentas, para realizar as análises estatísticas necessárias para a montagem da base de categorias: teste gráfico e teste do χ^2.
- Definição de regras dinâmicas para diversas linguagens de programação, oferecendo outras interfaces.
- Desenvolvimento de uma ligação para a ferramenta PSP2000, de modo que, juntas, venham a suprir o conjunto de necessidades do desenvolvedor no que se refere à utilização do PSP original e da instância proposta PSPi.

5.7 Considerações sobre o capítulo

Este capítulo apresentou a ferramenta gráfica PSPPlus-i que foi construída com o objetivo de apoiar a instância proposta por esta dissertação. As funcionalidades da ferramenta foram apresentadas utilizando dados reais do estudo de caso que será detalhado no próximo capítulo.

Essa ferramenta pode ser considerada como um protótipo, sujeito a novas implementações e melhorias, conforme será discutido no capítulo 7, na seção de trabalho futuros.

CAPÍTULO 6 - ESTUDO DE CASO

Como forma de validar as propostas efetuadas por esta dissertação, um estudo de caso foi conduzido, tendo sido realizado em duas etapas:

- A primeira etapa consistiu na execução do novo processo proposto pelo PSPi, o processo *PSPi0.2 Size Categories Base*, para a montagem da base de categorias de tamanho;
- A segunda etapa consistiu na aplicação da base de categorias de tamanho desenvolvida, em conjunto com as orientações para a concepção e estimativa de programas proposta através do *C36i - PROBEi Size Estimating Guidance.*

6.1 Ambiente de inserção

O estudo de caso desta dissertação foi realizado em um ambiente de grande porte de uma instituição financeira cuja porção *batch* dos sistemas de informação legados é, na sua maioria, codificada em linguagem COBOL, com ou sem acesso direto a informações em banco de dados utilizando SQL.

Os programas utilizados para a montagem da base de categorias de tamanho são provenientes de um sistema de financiamentos. Ao todo foram mensurados e catalogados 130 programas *batch* COBOL, sendo 100, da categoria Formatador e 30, da categoria Validador.

Os programas utilizados para a validação da base de categorias de tamanho são provenientes de um sistema de pagamentos do sistema financeiro. Ao todo foram estimados, mensurados e catalogados 25 programas *batch* desenvolvidos em COBOL, representando a totalidade dos programas *batch* da aplicação.

Como forma de reforçar a independência dos resultados, a segunda etapa do estudo de caso, validação da base de categorias de tamanho e validação do guia para apoiar as estimativas, foi conduzida por um profissional de mercado. Essas estimativas

foram feitas por uma analista de sistemas sênior, com larga experiência em sistemas de grande porte.

6.2 Primeira etapa: Montagem da base de categorias de tamanho

A base de categorias de tamanho foi montada pela autora dessa dissertação, seguindo o processo *PSPi0.2*, descrito na Tabela 14, localizada no capítulo 4, e utilizando como apoio a ferramenta PSPPlus-i. Foram executadas as seguintes atividades, que serão descritas nas próximas subseções:

- Identificação dos tipos de programas;
- Categorização dos programas selecionados;
- Mensuração dos programas selecionados;
- Análise da base de programas mensurados;
- Montagem das categorias de tamanho dos *proxies*.

6.2.1 Identificação dos tipos de programas

O ambiente do estudo de caso, conforme descrito no início deste capítulo, é o ambiente *mainframe* de aplicações comerciais de grande porte e, portanto, possui os mesmos tipos de programas que foram identificados na elaboração do modelo. Isso significa que o ambiente possui programas COBOL dos tipos: Receptor, Formatador e Atualizador.

Os programas do tipo Receptor utilizam linguagem de quarta geração proprietária do banco de dados e linguagem COBOL associada ao monitor transacional CICS, no caso, COBOL CICS. Conforme já abordado no capítulo 4, esse tipo de programa não será tratado por esta dissertação. Estão na mesma categoria, e, portanto, também não serão tratados, os programas do tipo Formatador on-line, ou seja, aqueles que consultam informações da aplicação e as disponibilizam na forma de relatórios on-line.

6.2.2 Seleção do conjunto de programas

A primeira atividade para a montagem da base de estimativas foi a seleção dos programas que a comporiam. Para garantir a representatividade dos programas

selecionados em relação ao universo que se estaria propondo atender, foram seguidos os critérios estabelecidos:

- A aplicação deveria pertencer ao segmento da linha de frente de negócios da empresa, refletindo o domínio de aplicações escolhido (no caso, o domínio de aplicações do setor financeiro);
- A parte *batch* do sistema deveria ser composta, em sua maioria, por programas desenvolvidos em COBOL, garantindo que as diversas categorias de programa estariam sendo representadas na amostra;
- O sistema deveria ter uma idade tal de modo a não distanciar as práticas utilizadas na época do desenvolvimento, das práticas atualmente em vigor na empresa.

Obedecendo aos critérios previamente definidos, foi selecionado um sistema do segmento de empréstimos que atende ao ramo dos financiamentos de máquinas e equipamentos agro-industriais. Este sistema é composto por uma parte on-line, desenvolvida em linguagem de quarta geração proprietária do banco de dados, e uma parte *batch*, desenvolvida quase que na sua totalidade em linguagem COBOL, com apenas alguns programas desenvolvidos em linguagem *Easytrieve* (menos de 5%). Por não fazer parte do escopo desta dissertação, os programas que compõem a parte on-line do sistema foram desprezados, bem como a parte *batch* desenvolvida em *Easytrieve*. As práticas de programação utilizadas são coerentes com as atualmente em vigor na empresa.

Ao todo foram selecionados 100 (cem) programas da categoria Formatador e 30 (trinta) programas da categoria Atualizador. Foram desprezados apenas os programas eventuais, que não fazem parte das rotinas produtivas do sistema, e que foram desenvolvidos para atender a uma demanda pontual, pois as práticas utilizadas nessas circunstâncias, nem sempre refletem as práticas rotineiras. Pode-se portanto, para fins estatísticos, considerar que se está trabalhando com toda a população de cada uma das categorias.

6.2.3 Mensuração dos programas selecionados

Os programas selecionados, e classificados de acordo com os tipos definidos Formatador e Atualizador, foram mensurados utilizando o módulo de mensuração da

ferramenta PSPPlus-i desenvolvida por esta dissertação e descrita no capítulo anterior. As regras de contagem estabelecidas na Tabela 13 do capítulo 4 foram obedecidas, tendo sido contadas as LOC úteis da *Procedure Division*.

Apesar de não serem utilizadas no primeiro momento, outras informações também foram obtidas dos programas selecionados e foram armazenadas na base de dados da ferramenta. Essas informações adicionais poderão ser utilizadas em trabalhos futuros como forma de refinar o modelo proposto e são:

- LOC úteis da *Identification Division*, da *Environment Division*, da *File Section* e da *Working Storage Section*;
- LOC em branco da *Identification Division*, da *Environment Division*, da *File Section* e da *Working Storage Section*, além das LOC em branco da própria *Procedure Division*;
- LOC úteis da *Identification Division*, da *Environment Division*, da *File Section* e da *Working Storage Section*;
- Quantidade de arquivos;
- Quantidade de tabelas de banco de dados (expressa na forma de declaração de *dataviews*);
- Quantidade de variáveis definidas na *Working Storage Section*;
- Quantidade de *Copys*.

O resultado da mensuração efetuada nos programas da categoria Formatador encontram-se detalhados na Tabela 21[47]. Conforme se pode observar pelos dados apresentados, referentes aos 100 programas utilizados:

- O tamanho total da *Procedure Division* variou entre 121 LOC e 2.399 LOC (esse tamanho, conforme especificado nas regras de contagem, refere-se às linhas úteis da *Procedure Division*, desconsiderando linhas em branco e linhas de comentário);
- Os programas possuíam entre 6 e 49 *sections*, também consideradas dentro da *Procedure Division*;

[47] A notação para os números apresentados nas tabelas deste capítulo seguem o padrão brasileiro, no qual o caractere "," indica decimais. Isto significa que 12,929 representa aproximadamente 13 LOC/*section* e não 13 KLOC/*section*.

- A densidade do código variou entre aproximadamente 13 a 70 LOC/*section*.

Num	*Loc Proc*	Qtde *Section*	*Loc/ Section*
1	181	14	12,929
2	232	16	14,500
3	196	13	15,077
4	175	11	15,909
5	240	15	16,000
6	190	11	17,273
7	121	7	17,286
8	143	8	17,875
9	363	19	19,105
10	275	14	19,643
11	275	14	19,643
12	334	17	19,647
13	180	9	20,000
14	260	13	20,000
15	201	10	20,100
16	262	13	20,154
17	323	16	20,188
18	223	11	20,273
19	268	13	20,615
20	313	15	20,867
21	238	11	21,636
22	261	12	21,750
23	291	13	22,385
24	565	25	22,600
25	704	31	22,710
26	570	25	22,800
27	274	12	22,833
28	231	10	23,100
29	233	10	23,300
30	305	13	23,462
31	290	12	24,167
32	171	7	24,429
33	421	17	24,765
34	772	31	24,903
35	374	15	24,933
36	710	28	25,357
37	407	16	25,438
38	229	9	25,444
39	154	6	25,667
40	316	12	26,333
41	396	15	26,400
42	273	10	27,300
43	356	13	27,385
44	674	24	28,083
45	309	11	28,091
46	281	10	28,100
47	285	10	28,500
48	658	23	28,609
49	745	26	28,654
50	491	17	28,882
51	261	9	29,000
52	551	19	29,000
53	322	11	29,273
54	506	17	29,765
55	360	12	30,000
56	367	12	30,583
57	184	6	30,667
58	404	13	31,077
59	530	17	31,176
60	787	25	31,480
61	381	12	31,750
62	796	25	31,840
63	416	13	32,000
64	417	13	32,077
65	451	14	32,214
66	873	27	32,333
67	393	12	32,750
68	820	25	32,800
69	197	6	32,833
70	908	27	33,630
71	506	15	33,733
72	375	11	34,091
73	721	21	34,333
74	654	19	34,421
75	792	23	34,435
76	554	16	34,625
77	877	25	35,080
78	319	9	35,444
79	610	17	35,882
80	845	23	36,739
81	486	13	37,385
82	416	11	37,818
83	736	19	38,737
84	1144	29	39,448
85	1223	31	39,452
86	847	21	40,333
87	446	11	40,545
88	752	18	41,778
89	397	9	44,111
90	1039	23	45,174
91	853	18	47,389
92	2399	49	48,959
93	1134	23	49,304
94	1125	22	51,136
95	1386	27	51,333
96	1242	24	51,750
97	316	6	52,667
98	739	13	56,846
99	1669	28	59,607
100	983	14	70,214

Tabela 21. Mensuração dos programas da categoria Formatador.

A mensuração efetuada nos programas do tipo Atualizador encontram-se descritas na Tabela 22 e o que se pode observar entre os 30 programas utilizados é que:

- O tamanho dos programas variou entre 68 e 3.248 LOC;
- Os programas foram codificados utilizando entre 4 e 43 *sections*;
- A densidade do código variou entre aproximadamente 11 a 75 LOC/*section.*

Num	*Loc Proc*	Qtde *Section*	*Loc/ Section*	Num	*Loc Proc*	Qtde *Section*	*Loc/ Section*
1	68	6	11,333	16	169	6	28,167
2	89	7	12,714	17	282	10	28,200
3	105	7	15,000	18	692	24	28,833
4	107	7	15,286	19	295	10	29,500
5	68	4	17,000	20	174	5	34,800
6	156	9	17,333	21	441	12	36,750
7	310	17	18,235	22	232	6	38,667
8	132	7	18,857	23	556	14	39,714
9	197	10	19,700	24	883	22	40,136
10	170	8	21,250	25	615	14	43,929
11	85	4	21,250	26	495	10	49,500
12	186	8	23,250	27	697	14	49,786
13	202	8	25,250	28	844	16	52,750
14	1145	43	26,628	29	348	5	69,600
15	221	8	27,625	30	3248	43	75,535

Tabela 22. Mensuração dos programas da categoria Atualizador.

Apesar da diferença entre tamanhos, quantidade de *sections* e LOC/*section* dos extremos das duas categorias não parecer significativa, optou-se por manter a condução da análise de cada uma das categorias separadamente.

6.2.4 Análise da base de programas da categoria Formatador

Conforme citado no capítulo 4, a maioria dos procedimentos utilizados nas estimativas de tamanho de programa e gerenciamento da qualidade de software envolvendo métodos estatísticos parte do princípio que os dados seguem a distribuição normal [HUMPHREY, 1995]. Por isso, para cada uma das categorias de programas, Atualizador e Formatador, foram realizados os dois testes sugeridos:

- O teste da plotagem do gráfico e posterior análise visual;
- O teste do χ^2.

As equações estatísticas empregadas estão descritas na Tabela 15, apresentada no capítulo 4, bem como os parâmetros necessários para a sua utilização. As análises foram realizadas sobre as informações de LOC/*section* e conduzidas separadamente para cada uma das categorias. Os testes executados sobre os programas da categoria Formatador serão discutidos nesta seção e os testes realizados sobre os programas da categoria Atualizador, na próxima seção.

Na primeira análise realizada, a visual, baseada na plotagem dos valores obtidos na mensuração dos programas da categoria Formatador, contra os valores da distribuição normal, utilizou-se os procedimentos descritos na Tabela 16. Esses procedimentos consistem, basicamente, em comparar como se comportam as duas distribuições, a normal e a que está sendo analisada, quando apresentadas graficamente através de sua função de densidade cumulativa (CDF)[48].

Os dados que caracterizam a distribuição das LOC/*section* da categoria Formatador foram obtidos aplicando-se a Formula 1, a Formula 2 e a Formula 3, gerando respectivamente:

- Média : $x_{avg} = 30{,}441$;
- Variância : $\sigma^2 = 112{,}347$; e
- Desvio padrão: $\sigma = 10{,}599$.

Para apoiar a plotagem do gráfico da CDF foram elaboradas duas tabelas:

- a Tabela 23, que contém os termos normalizados (z_i) dos 100 programas da categoria Formatador, calculados usando a Formula 4; e
- a Tabela 24, que contém a CDF da distribuição normal, considerando intervalos de desvio padrão de –4,0 a +4,0, incrementados de 0,2 em 0,2, e a CDF dos dados que estão sendo testados.

Os valores da CDF da distribuição normal podem ser obtidos diretamente de tabelas estatísticas ou podem ser calculados utilizando ferramentas estatísticas. Como a ferramenta PSPPlus-i ainda não dispõe de um módulo estatístico, os valores aqui apresentados foram calculados utilizando a função DIST.NORMP do Microsoft Excel.

Os valores da CDF dos dados da categoria Formatador foram obtidos comparando-se os termos normalizados *z,* da distribuição normal, com os termos normalizados z_i, dos dados que estão sendo analisados.

[48] Do original, em inglês, *cumulative density function (CDF).*

Num	*Loc/ Section*	i/n	zi
1	12,929	0,01	-1,652
2	14,500	0,02	-1,504
3	15,077	0,03	-1,450
4	15,909	0,04	-1,371
5	16,000	0,05	-1,362
6	17,273	0,06	-1,242
7	17,286	0,07	-1,241
8	17,875	0,08	-1,186
9	19,105	0,09	-1,069
10	19,643	0,10	-1,019
11	19,643	0,11	-1,019
12	19,647	0,12	-1,018
13	20,000	0,13	-0,985
14	20,000	0,14	-0,985
15	20,100	0,15	-0,976
16	20,154	0,16	-0,971
17	20,188	0,17	-0,967
18	20,273	0,18	-0,959
19	20,615	0,19	-0,927
20	20,867	0,20	-0,903
21	21,636	0,21	-0,831
22	21,750	0,22	-0,820
23	22,385	0,23	-0,760
24	22,600	0,24	-0,740
25	22,710	0,25	-0,729
26	22,800	0,26	-0,721
27	22,833	0,27	-0,718
28	23,100	0,28	-0,693
29	23,300	0,29	-0,674
30	23,462	0,30	-0,658
31	24,167	0,31	-0,592
32	24,429	0,32	-0,567
33	24,765	0,33	-0,536
34	24,903	0,34	-0,522
35	24,933	0,35	-0,520
36	25,357	0,36	-0,480
37	25,438	0,37	-0,472
38	25,444	0,38	-0,471
39	25,667	0,39	-0,450
40	26,333	0,40	-0,388
41	26,400	0,41	-0,381
42	27,300	0,42	-0,296
43	27,385	0,43	-0,288
44	28,083	0,44	-0,222
45	28,091	0,45	-0,222
46	28,100	0,46	-0,221
47	28,500	0,47	-0,183
48	28,609	0,48	-0,173
49	28,654	0,49	-0,169
50	28,882	0,50	-0,147
51	29,000	0,51	-0,136
52	29,000	0,52	-0,136
53	29,273	0,53	-0,110
54	29,765	0,54	-0,064
55	30,000	0,55	-0,042
56	30,583	0,56	0,013
57	30,667	0,57	0,021
58	31,077	0,58	0,060
59	31,176	0,59	0,069
60	31,480	0,60	0,098
61	31,750	0,61	0,123
62	31,840	0,62	0,132
63	32,000	0,63	0,147
64	32,077	0,64	0,154
65	32,214	0,65	0,167
66	32,333	0,66	0,179
67	32,750	0,67	0,218
68	32,800	0,68	0,223
69	32,833	0,69	0,226
70	33,630	0,70	0,301
71	33,733	0,71	0,311
72	34,091	0,72	0,344
73	34,333	0,73	0,367
74	34,421	0,74	0,375
75	34,435	0,75	0,377
76	34,625	0,76	0,395
77	35,080	0,77	0,438
78	35,444	0,78	0,472
79	35,882	0,79	0,513
80	36,739	0,80	0,594
81	37,385	0,81	0,655
82	37,818	0,82	0,696
83	38,737	0,83	0,783
84	39,448	0,84	0,850
85	39,452	0,85	0,850
86	40,333	0,86	0,933
87	40,545	0,87	0,953
88	41,778	0,88	1,070
89	44,111	0,89	1,290
90	45,174	0,90	1,390
91	47,389	0,91	1,599
92	48,959	0,92	1,747
93	49,304	0,93	1,780
94	51,136	0,94	1,952
95	51,333	0,95	1,971
96	51,750	0,96	2,010
97	52,667	0,97	2,097
98	56,846	0,98	2,491
99	59,607	0,99	2,752
100	70,214	1,00	3,752

Tabela 23. Termos normalizados da categoria Formatador.

VALOR Z DISTRIBUIÇÃO NORMAL	CDF DA DISTRIBUIÇÃO NORMAL	CDF DAS LOC/SECTION ATUALIZADOR
-4	0,0000	0
-3,8	0,0001	0
-3,6	0,0002	0
-3,4	0,0003	0
-3,2	0,0007	0
-3	0,0013	0
-2,8	0,0026	0
-2,6	0,0047	0
-2,4	0,0082	0
-2,2	0,0139	0
-2	0,0228	0
-1,8	0,0359	0
-1,6	0,0548	0
-1,4	0,0808	0
-1,2	0,1151	0,04
-1	0,1587	0,08
-0,8	0,2119	0,24
-0,6	0,2743	0,32
-0,4	0,3446	0,38
-0,2	0,4207	0,5
0	0,5000	0,56
0,2	0,5793	0,66
0,4	0,6554	0,76
0,6	0,7257	0,82
0,8	0,7881	0,86
1	0,8413	0,88
1,2	0,8849	0,9
1,4	0,9192	0,9
1,6	0,9452	0,9
1,8	0,9641	0,92
2	0,9772	0,96
2,2	0,9861	0,96
2,4	0,9918	0,96
2,6	0,9953	0,98
2,8	0,9974	0,98
3	0,9987	0,98
3,2	0,9993	0,98
3,4	0,9997	0,98
3,6	0,9998	0,98
3,8	0,9999	1
4	1,0000	1

Tabela 24. CDF Distribuição normal x CDF da categoria Formatador.

A Figura 22 mostra o gráfico da CDF das duas distribuições. Nele se pode comparar visualmente a curva da CDF da distribuição normal com a curva da CDF dos programas da categoria Formatador.

Como se pode observar, as duas curvas se aproximam, com pequenas divergências em dois segmentos. Enquanto na distribuição normal a curva desliza suavemente ao longo do eixo x, nos dados que estão sendo analisados, a curva tem uma queda brusca, cortando o eixo x. Apesar disto, pode-se considerar que os dados da categoria Formatador se aproximam da distribuição normal.

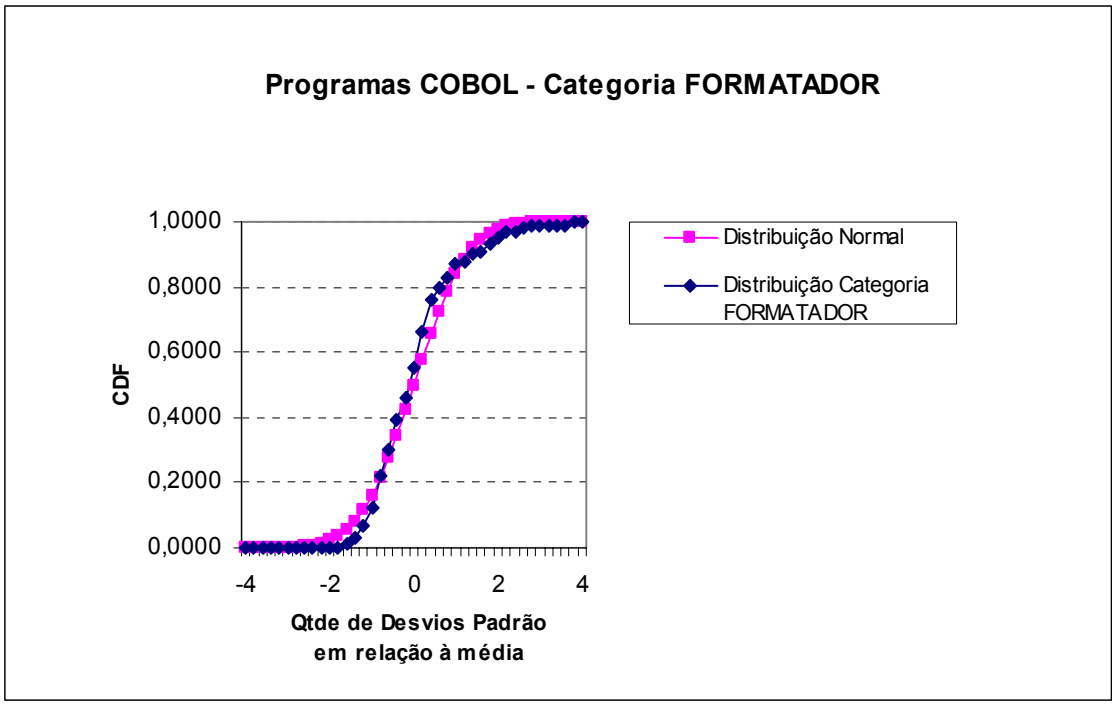

Figura 22. Resultado do teste gráfico – categoria Formatador.

Constata-se, ainda, pela Figura 22, que a distribuição se concentra entre –2,0 e +2,0 desvios padrões em torno da média o que significa que pode-se utilizar os critérios para a montagem das categorias de tamanho relativo estabelecidos no capítulo 4.

Para confirmar a conclusão obtida graficamente, foi conduzido o teste do χ^2. Para a realização desse teste é recomendado que se tenha 30 pontos ou mais

[HUMPHREY, 1995]. Como a base da categoria Formatador contém 100 elementos, pode-se dizer que o teste apresenta um grau de confiabilidade adequado.

Os procedimentos utilizados estão descritos na Tabela 17 e as equações referenciadas nesses procedimentos são as mesmas já apresentadas na Tabela 15.

Primeiramente, optou-se por dividir a distribuição normal em 10 segmentos (S = 10) que é o valor que se encaixa dentro das condições recomendadas:

- Resulta em um número de elementos por segmento (N) inteiro e maior ou igual a cinco (N = n/S = 100/10 = 10);
- S é maior do que 3 (S=10); e
- S^2 é maior ou igual a n ($S^2=100=n$).

Os valores dos limites superior e inferior foram obtidos na tabela de segmentos da distribuição normal, disponível em [HUMPHREY, 1995], Apêndice A, para S = 10. Os demais valores da Tabela 25 foram calculados para auxiliar a obtenção do fator Q, conforme Formula 5 da Tabela 15.

SEGMENTO	LIMITE INFERIOR	LIMITE SUPERIOR	Q. ITENS NORMAIS *Ni*	Q. ITENS DE DADOS *Ki*	VALORES DE $(Ni-Ki)^2/Ni$
1	-∞	-1,2820	10	5	2,5
2	-1,2820	-0,8420	10	15	2,5
3	-0,8420	-0,5240	10	13	0,9
4	-0,5240	-0,2530	10	10	0
5	-0,2530	0,0000	10	12	0,4
6	0,0000	0,2530	10	14	1,6
7	0,2530	0,5240	10	10	0
8	0,5240	0,8420	10	4	3,6
9	0,8420	1,2820	10	5	2,5
10	1,2820	+∞	10	12	0,4
					Q = 14,4

Tabela 25. Resultado do teste do χ^2 – categoria Formatador.

O valor da probabilidade pode ser obtido consultando a tabela estatística de χ^2 para $n-1$ graus de liberdade, ou utilizando a fórmula matemática da distribuição de χ^2 ou utilizando ferramentas com funções estatísticas. No caso, utilizou-se a função estatística

do Microsoft Excel DIST.QUI, para 9 graus de liberdade, obtendo um valor de (1-p) = 0,10879.

Valores iguais ou inferiores a 0,005 indicam que a distribuição pode ser seguramente considerada como não próxima da distribuição normal. Valores iguais ou superiores a 0,2 indicam que a distribuição seguramente se aproxima da distribuição normal. Valores intermediários indicam graus variados de confiabilidade em relação à aproximação normal [HUMPHREY, 1995]. Como o valor obtido encontra-se na faixa intermediária, isto confirma o que foi observado visualmente no gráfico apresentado na Figura 22, ou seja, a distribuição dos dados da categoria Formatador se aproxima da distribuição normal.

6.2.5 Análise da base de programas da categoria Atualizador

Analogamente aos programas da categoria Formatador, os programas da categoria Atualizador também foram submetidos aos dois testes:

- O teste da plotagem do gráfico e posterior análise visual;
- O teste do χ^2.

As equações estatísticas empregadas foram as mesmas descritas na Tabela 15, apresentada no capítulo 4 e as análises foram realizadas sobre as informações de LOC/*section*. Para isso, inicialmente foram calculados os dados que caracterizam a distribuição das LOC/*section* dessa categoria. Ao aplicar a Formula 1, a Formula 2 e a Formula 3, obteve-se, respectivamente:

- Média : $x_{avg} = 31,219$;
- Variância : $\sigma^2 = 248,48$; e
- Desvio padrão: $\sigma = 15,763$.

Apesar das duas distribuições possuírem médias parecidas, a da categoria Formatador é 30,441 e a da categoria Atualizador, 31,219, elas possuem variância e desvio padrão bastante diferentes. Enquanto a variância da categoria Formatador é de 112,347, a da categoria Atualizador é mais do que duas vezes maior, 248,48. Da mesma forma, o desvio padrão é cerca de 50% maior na categoria Atualizador, sendo igual a 15,763, enquanto que o da categoria Atualizador é de 10,599. Isso significa que os dados da categoria Formatador se encontram mais concentrados em torno da média do que os programas da categoria Atualizador.

Para auxiliar a montagem das curvas da CDF da distribuição normal e da distribuição dos programas da categoria Atualizador foram desenvolvidas:

- a Tabela 26, que contém os termos normalizados (z_i) dos 30 programas da categoria Atualizador, calculados usando a Formula 4; e
- a Tabela 27, que contém a CDF da distribuição normal, considerando intervalos de desvio padrão de –4,0 a +4,0, incrementados de 0,2 em 0,2, e a CDF dos dados que estão sendo testados.

Novamente ressalta-se que os valores da CDF da distribuição normal podem ser obtidos diretamente de tabelas estatísticas ou podem ser calculados utilizando ferramentas estatísticas. Os valores aqui apresentados foram calculados utilizando a função DIST.NORMP do Microsoft Excel. Os valores da CDF dos dados da categoria Atualizador foram obtidos comparando-se os termos normalizados *z,* da distribuição normal, com os termos normalizados z_i, dos dados que estão sendo analisados.

Num	**LOC/ *Section***	***i/n***	***zi***	**Num**	**LOC/ *Section***	***i/n***	***zi***
1	11,333	0,0333	-1,262	16	28,167	0,5333	-0,194
2	12,714	0,0667	-1,174	17	28,200	0,5667	-0,192
3	15,000	0,1000	-1,029	18	28,833	0,6000	-0,151
4	15,286	0,1333	-1,011	19	29,500	0,6333	-0,109
5	17,000	0,1667	-0,902	20	34,800	0,6667	0,227
6	17,333	0,2000	-0,881	21	36,750	0,7000	0,351
7	18,235	0,2333	-0,824	22	38,667	0,7333	0,472
8	18,857	0,2667	-0,784	23	39,714	0,7667	0,539
9	19,700	0,3000	-0,731	24	40,136	0,8000	0,566
10	21,250	0,3333	-0,632	25	43,929	0,8333	0,806
11	21,250	0,3667	-0,632	26	49,500	0,8667	1,160
12	23,250	0,4000	-0,506	27	49,786	0,9000	1,178
13	25,250	0,4333	-0,379	28	52,750	0,9333	1,366
14	26,628	0,4667	-0,291	29	69,600	0,9667	2,435
15	27,625	0,5000	-0,228	30	75,535	1,0000	2,811

Tabela 26. Termos normalizados da categoria Atualizador.

VALOR Z DISTRIBUIÇÃO NORMAL	CDF DA DISTRIBUIÇÃO NORMAL	CDF DAS LOC/*SECTION* ATUALIZADOR
-4	0,0000	0
-3,8	0,0001	0
-3,6	0,0002	0
-3,4	0,0003	0
-3,2	0,0007	0
-3	0,0013	0
-2,8	0,0026	0
-2,6	0,0047	0
-2,4	0,0082	0
-2,2	0,0139	0
-2	0,0228	0
-1,8	0,0359	0
-1,6	0,0548	0
-1,4	0,0808	0
-1,2	0,1151	0,0333
-1	0,1587	0,1333
-0,8	0,2119	0,2333
-0,6	0,2743	0,3667
-0,4	0,3446	0,4
-0,2	0,4207	0,5
0	0,5000	0,6333
0,2	0,5793	0,6333
0,4	0,6554	0,7
0,6	0,7257	0,8
0,8	0,7881	0,8
1	0,8413	0,8333
1,2	0,8849	0,8667
1,4	0,9192	0,9333
1,6	0,9452	0,9333
1,8	0,9641	0,9333
2	0,9772	0,9333
2,2	0,9861	0,9333
2,4	0,9918	0,9333
2,6	0,9953	0,9667
2,8	0,9974	0,9667
3	0,9987	1
3,2	0,9993	1
3,4	0,9997	1
3,6	0,9998	1
3,8	0,9999	1
4	1,0000	1

Tabela 27. CDF Distribuição normal x CDF da categoria Atualizador.

A Figura 23 apresenta as curvas obtidas ao plotar a CDF da distribuição normal e a CDF das informações de LOC/*section* dos programas da categoria Atualizador. Analogamente à categoria Formatador, a categoria Atualizador também se aproxima da distribuição normal, ao ser analisada visualmente.

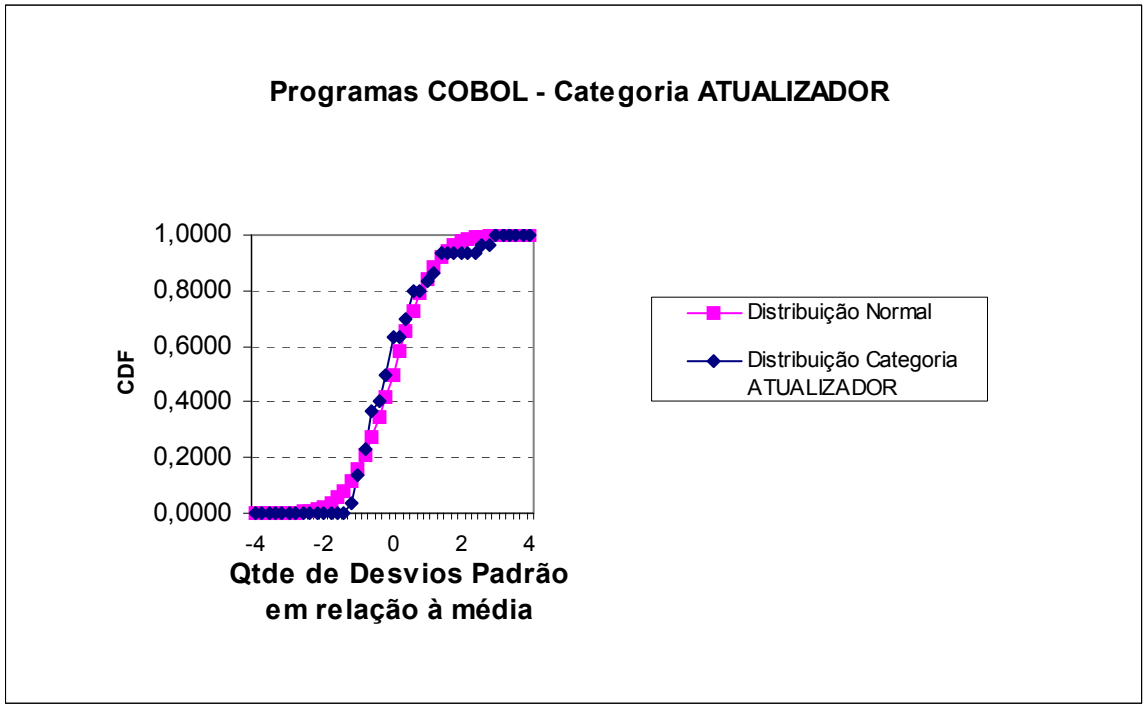

Figura 23. Resultado do teste gráfico - categoria Atualizador.

Em seguida, como forma de confirmar os resultados observados graficamente, foi realizado o teste do χ^2. Conforme já havia sido dito na seção anterior, para a realização desse teste é recomendado que se tenha 30 pontos ou mais [HUMPHREY, 1995]. Como a base da categoria Atualizador contém exatamente 30 elementos, pode-se dizer que se está no limite mínimo de condições para realizar o teste e que seu resultado deve ser analisado com cuidado.

Optou-se por dividir a distribuição normal em 6 segmentos ($S = 6$) cujo valor se encaixa dentro das condições recomendadas:

- Resulta em um número de elementos por segmento (N) inteiro e maior ou igual a cinco ($N = n/S = 30/6 = 5$);

- *S* é maior do que 3 (*S*=6); e
- S^2 é maior ou igual a *n* (S^2=36) >(*n*=30).

Os valores dos limites superior e inferior foram obtidos na tabela de segmentos da distribuição normal disponível em [HUMPHREY, 1995], Apêndice A, para *S* = 6. Os demais valores da Tabela 28 foram calculados para auxiliar a obtenção do fator *Q*, conforme Formula 5 da Tabela 15.

SEGMENTO	LIMITE INFERIOR	LIMITE SUPERIOR	Q. ITENS NORMAIS *Ni*	Q. ITENS DE DADOS *Ki*	VALORES DE $(Ni-Ki)^2/Ni$
1	-∞	-0,9670	5	4	0,2
2	-0,9670	-0,4310	5	8	1,8
3	-0,4310	0,0000	5	7	0,8
4	0,0000	0,4310	5	2	1,8
5	0,4310	0,9670	5	4	0,2
6	0,9670	+∞	5	5	0
					Q = 4,8

Tabela 28. Resultado do teste do χ^2 - categoria Atualizador.

Assim como no teste do χ^2 realizado para os programas do tipo Formatador, o valor da probabilidade foi obtido utilizando a função estatística do Microsoft Excel DIST.QUI, para 5 graus de liberdade, obtendo um valor de (1-p) = 0,44077.

Conforme citado anteriormente, valores iguais ou superiores a 0,2 indicam que a distribuição seguramente se aproxima da distribuição normal. Logo, conclui-se que a distribuição dos valores de LOC/*section* da categoria Atualizador se aproxima da distribuição normal.

6.2.6 Montagem das categorias de tamanho dos *proxies*

Conforme os testes que foram realizados, e que se encontram descritos nas duas seções anteriores, ambas distribuições de dados das categorias de tamanho se aproximam da distribuição normal. Isso significa que se pode utilizar os conceitos apresentados no capítulo 4 para a montagem das tabelas dos *proxies,* ou seja, trabalhar com os valores que caracterizam as distribuições, a média (μ) e o desvio padrão (σ), da seguinte forma:

- Muito pequeno (VS) : $\mu - 2\sigma$
- Pequeno (S) : $\mu - \sigma$
- Médio (M) : μ
- Grande (L) : $\mu + \sigma$
- Muito grande (VL) : $\mu + 2\sigma$

A Tabela 29 apresenta as categorias de tamanho relativo dos *proxies* para estimar os programas do tipo Formatador, expressas em LOC/*section.* Conforme se pode observar, os valores parecem consistentes, considerando que o valor mínimo dos dados da amostra dessa categoria era aproximadamente 13 LOC/*section* e o valor máximo, aproximadamente 70 LOC/*section*, sendo esse último o único ponto acima de 59 LOC/*section.*

CATEGORIAS DE TAMANHO PARA COBOL - FORMATADOR (LOC/*SECTION*)					
	MUITO PEQUENO (VS)	**PEQUENO (S)**	**MÉDIO (M)**	**GRANDE (L)**	**MUITO GRANDE (VL)**
	$\mu - 2 * \sigma$	**$\mu - \sigma$**	**μ**	**$\mu + \sigma$**	**$\mu + 2 * \sigma$**
FORMATADOR	9,242	19,842	30,441	41,041	51,640

Tabela 29. Categorias de tamanho relativo – COBOL – categoria Formatador.

Ao aplicar essas mesmas regras para a construção da tabela para os programas do tipo Atualizador, no entanto, os resultados não foram tão satisfatórios, conforme se observa nos valores apresentados na Tabela 30. O valor para a categoria muito pequeno (VS) ficou negativo, o que, certamente, é um valor impossível.

CATEGORIAS DE TAMANHO PARA COBOL - ATUALIZADOR (LOC/*SECTION*)					
	MUITO PEQUENO (VS)	**PEQUENO (S)**	**MÉDIO (M)**	**GRANDE (L)**	**MUITO GRANDE (VL)**
	$\mu - 2 * \sigma$	**$\mu - \sigma$**	**μ**	**$\mu + \sigma$**	**$\mu + 2 * \sigma$**
ATUALIZADOR	-0,307	15,456	31,219	46,983	62,746

Tabela 30. Categorias de tamanho relativo - COBOL - categoria Atualizador - 1ª tentativa.

Decidiu-se, então, aplicar a orientação fornecida em [HUMPHREY, 1995], fazendo a transformação dos termos através de seu logaritmo neperiano. Essa técnica consiste, basicamente, em se calcular o logaritmo neperiano das informações de LOC/*section* e trabalhar com o valor transformado até o momento do cálculo das categorias, quando, então, se retorna ao valor LOC/*section*. O resultado dessa transformação aplicada aos tamanhos dos elementos da categoria Atualizador é apresentado na Tabela 31.

Num	**LOC/ *Section***	***ln* *(LOC/Section)***	**Num**	**LOC/ *Section***	***ln* *(LOC/Section)***
1	11,333	2,4277	16	28,167	3,3381
2	12,714	2,5427	17	28,200	3,3393
3	15,000	2,7081	18	28,833	3,3615
4	15,286	2,7269	19	29,500	3,3844
5	17,000	2,8332	20	34,800	3,5496
6	17,333	2,8526	21	36,750	3,6041
7	18,235	2,9034	22	38,667	3,6550
8	18,857	2,9369	23	39,714	3,6817
9	19,700	2,9806	24	40,136	3,6923
10	21,250	3,0564	25	43,929	3,7826
11	21,250	3,0564	26	49,500	3,9020
12	23,250	3,1463	27	49,786	3,9077
13	25,250	3,2288	28	52,750	3,9656
14	26,628	3,2820	29	69,600	4,2428
15	27,625	3,3187	30	75,535	4,3246

Tabela 31. Aplicação de *ln* na categoria Atualizador.

A partir dos novos elementos, obtidos com a transformação sugerida, foram recalculados os novos valores[49] da média, variância e desvio padrão, utilizando as equações da Tabela 15:

- Média : $x_{avg} = 3{,}324$;
- Variância : $\sigma^2 = 0{,}238$; e
- Desvio padrão: $\sigma = 0{,}488$.

A partir da média e do desvio padrão foram efetuados os cálculos para obter o tamanho relativo das categorias e, em seguida, aplicada a transformação inversa, usando

[49] Note que esses valores estão considerando os termos transformados através de seu logaritmo neperiano.

o número neperiano *e*. O resultado são as categorias de tamanho relativo dos *proxies* dos programas do tipo Atualizador, apresentados na Tabela 30.

CATEGORIAS DE TAMANHO PARA COBOL - ATUALIZADOR (LOC/*SECTION*)					
	MUITO PEQUENO (VS)	**PEQUENO (S)**	**MÉDIO (M)**	**GRANDE (L)**	**MUITO GRANDE (VL)**
	$\mu - 2 * \sigma$	**$\mu - \sigma$**	**μ**	**$\mu + \sigma$**	**$\mu + 2 * \sigma$**
ATUALIZADOR	10,479	17,062	27,782	45,238	73,661

Tabela 32. Categorias de tamanho relativo - COBOL - categoria Atualizador.

Como se pode observar, os valores agora são mais coerentes com os elementos da amostra e não aparece mais o valor negativo para a categoria muito pequeno (VS).

6.3 Segunda etapa: Validação da base de categorias de tamanho

Uma vez montadas as duas bases de categorias de tamanho relativo de programa COBOL, é necessário validar se as orientações propostas por esta dissertação, combinadas com essas bases, resultam em um bom método de estimativas.

Conforme esclarecido no início deste capítulo, de modo a reforçar a independência dos resultados, essa validação foi efetuada por uma analista de sistemas sênior, com experiência em desenvolvimento e manutenção de sistemas no ambiente *mainframe.*

Foi fornecido para essa analista:

- O guia de estimativas *C36i – PROBEi Size Estimating Guidance* desenvolvido por esta dissertação;
- A base de categorias de tamanho relativo dos *proxies* dos programas do tipo Formatador, desenvolvida na primeira etapa do estudo de caso, utilizando as orientações propostas por esta dissertação através do processo PSPi0.2;
- A base de categorias de tamanho relativo dos *proxies* dos programas do tipo Atualizador, também desenvolvida na primeira etapa do estudo de caso, utilizando as orientações propostas por esta dissertação através do processo PSPi02;

- O formulário de estimativas *C39i Size Estimating Template* cujas complementações foram propostas por esta dissertação*;*
- A ferramenta PSPPlus-i, também desenvolvida por esta dissertação.

Essa segunda etapa da validação consistiu de duas atividades, que serão detalhadas nas próximas subseções:

- A analista de sistemas seguiu as orientações do guia *C36i – PROBEi Size Estimating Guidance,* utilizou as bases de categorias de tamanho relativo, e estimou o tamanho de cada um dos programas componentes do sistema de pagamentos;
- Os programas reais do sistema de pagamentos foram submetidos à ferramenta PSPPlus-i para que fosse determinado o número real de LOC e os resultados da estimativa foram comparados com esses dados reais.

6.3.1 Estimativa de tamanho utilizando o método proposto

Para iniciar as estimativas de tamanho a analista de sistemas dispunha, além de sua experiência em análise, das informações básicas sobre os programas, como: quantidade de arquivos de entrada e saída, quantidade de subrotinas acessadas, quantidade de regras de negócio e necessidade ou não de classificação dos dados. Ela não tinha experiência anterior em estimar LOC.

Primeiramente, a analista de sistemas identificou a quantidade de *sections* necessárias para cada programa e qual o tamanho relativo mais adequado de cada *section* (VS, S, M, L ou VL), registrando essas informações no formulário *C39i Size Estimating Template.* Em seguida, utilizando a base de categorias de tamanho relativo da categoria Formatador (Tabela 29, pág. 122) e da categoria Atualizador (Tabela 32, pág. 124), calculou o tamanho total do programa.

A Tabela 33 ilustra a estimativa realizada para um dos programas da categoria Formatador e a Tabela 34, a estimativa para um dos programas da categoria Atualizador. O programa ilustrado na Tabela 33 já havia sido apresentado no capítulo anterior, quando da apresentação da ferramenta PSPPlus-i (subseção Figura 18).

NEW OBJECTS	VS	S	M	L	VL
MAIN SECTION (S)		1			
STARTING SECTION (M)			1		
NORMAL ENDING SECTION (S)		1			
ABNORMAL ENDING SECTION (S)		1			
OPEN SECTION (S)		6			
CLOSE SECTION (S)		6			
READ SECTION - VSAM/SEQ (S)		1			
READ SECTION – DB (M)			4		
WRITE SECTION – VSAM/SEQ (S)		1			
WRITE SECTION – DB (M)			2		
EXTERNAL CALL SECTION (S to M)		4			
REPORT CONTROL SECTION (S)					
SELECTION SECTION (3/S)		12			
CALCULATION SECTION (5/S)		4			
UPDATE SECTION (15/S)					
CLASSIFICATION SECTION (VS to VL)					
OTHER SECTIONS (VS to VL)					
NUMBER OF SECTIONS	0	37	7	0	0
LOC/SECTION (Formatter Size Base)	9,24	19,84	30,44	41,04	51,64
TOTAL LOC/SECTION	0	734,08	213,08	0	0
TOTAL NEW OBJECTS (NO)	=>	=>	=>	=>	**947**

Tabela 33. Estimativa usando *C39i - Size Estimating Template* - Formatador.

NEW OBJECTS	VS	S	M	L	VL
MAIN SECTION (S)		1			
STARTING SECTION (M)			1		
NORMAL ENDING SECTION (S)		1			
ABNORMAL ENDING SECTION (S)		1			
OPEN SECTION (S)		5			
CLOSE SECTION (S)		5			
READ SECTION - VSAM/SEQ (S)		1			
READ SECTION – DB (M)			3		
WRITE SECTION – VSAM/SEQ (S)		1			
WRITE SECTION – DB (M)			1		
EXTERNAL CALL SECTION (S to M)		1			
REPORT CONTROL SECTION (S)					
SELECTION SECTION (3/S)		1			
CALCULATION SECTION (5/S)					
UPDATE SECTION (15/S)		2			
CLASSIFICATION SECTION (VS to VL)				1	1
OTHER SECTIONS (VS to VL)					
NUMBER OF SECTIONS	0	19	5	1	1
LOC/SECTION (Updater Size Base)	10,48	17,06	27,78	45,24	73,66
TOTAL LOC/SECTION	0	324,14	138,9	45,24	73,66
TOTAL NEW OBJECTS (NO)	=>	=>	=>	=>	**582**

Tabela 34. Estimativa usando *C39i - Size Estimating Template* - Atualizador.

Conforme dito no início deste capítulo, a aplicação utilizada para a validação das bases de categorias de tamanho é composta de 25 programas ao todo, sendo 15 do tipo Atualizador e 10 do tipo Formatador. As tabelas acima retratam apenas um exemplo de cada categoria. Os dados referentes aos demais programas estimados encontram-se sumarizados na próxima subseção.

6.3.2 Análise das estimativas

Uma vez concluída a etapa de estimativas, os programas foram submetidos à mensuração utilizando o módulo de mensuração da ferramenta desenvolvida PSPPlus-i. Conforme já havia sido discutido anteriormente, a ferramenta segue as regras estabelecidas no padrão de contagem de LOC (Tabela 13 - COBOL *LOC Counting Standard.*). Isso significa que foram contadas as LOC úteis da *Procedure Division.*

Os resultados obtidos com a mensuração dos programas da categoria Formatador são apresentados na Tabela 35. Conforme se pode observar, o método de estimativas para essa categoria tende a produzir estimativas a maior. Note que apenas um dos programas foi estimado a menor, o programa número 1, os demais, foram todos estimados a maior. Na realidade o programa número 1 possui características um pouco diferentes dos demais, uma vez que tem uma função de validação de campos bastante robusta, e que não foi considerada no momento da estimativa. Sendo assim, considerando os demais programas, o método parece induzir o desenvolvedor a produzir estimativas de tamanho superiores aos tamanhos reais.

NÚMERO PROGRAMA	LOC ESTIMADAS	LOC REAIS	% ERRO
1	1404	1762	25,53
2	511	418	-18,15
3	500	381	-23,82
4	470	387	-17,60
5	410	288	-29,78
6	288	182	-36,89
7	209	107	-48,81
8	947	883	-6,78
9	460	376	-18,34
10	279	168	-39,82

Tabela 35. Resultado da mensuração - categoria Formatador.

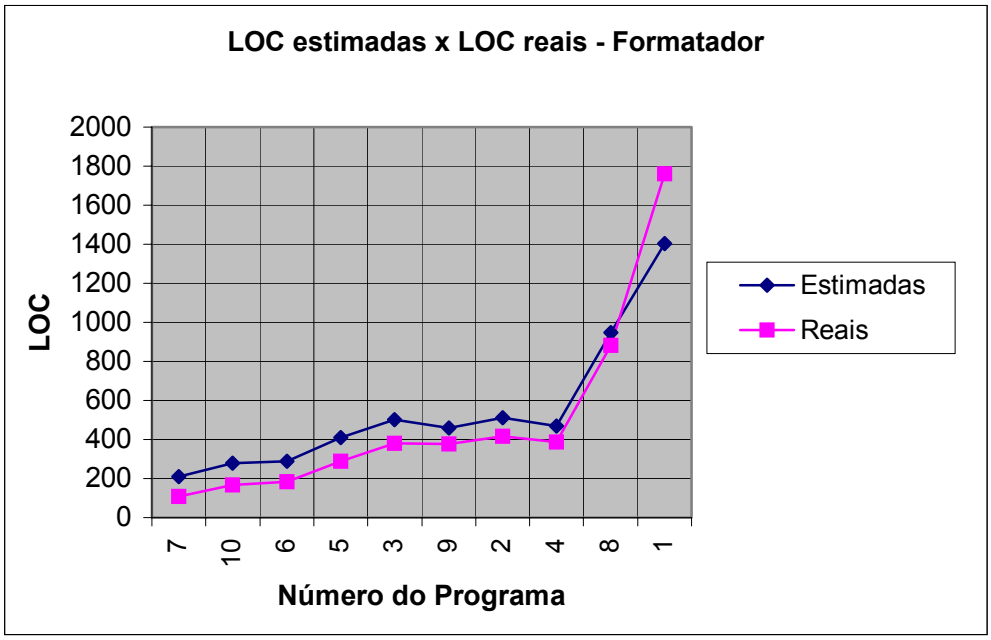

Figura 24. LOC estimadas x LOC reais - Formatador.

A Figura 24 ilustra, de forma gráfica, as mesmas informações contidas na Tabela 35, ou seja, compara as LOC estimadas com as LOC reais para cada um dos programas da categoria Formatador. Para facilitar a visualização, optou-se por classificar os elementos na ordem crescente do % de erro da estimativa.

A Figura 25 também apresenta a comparação entre as LOC estimadas e reais, só que na forma de gráfico de dispersão, incluindo a linha de tendência linear e a respectiva equação linear. Essa equação poderá ser utilizada posteriormente nas próximas estimativas, como forma de adequar os desvios naturais da estimativa. Como se pode observar no mesmo gráfico, a correlação entre as LOC estimadas e as LOC reais para os programas do tipo Formatador é bastante alta, $r^2 = 0,9789$.

A aplicação da regressão linear faz parte do método de estimativas PROBE, conforme já havia sido descrito no capítulo 3 e o fato da base apresentar uma correlação elevada indica que essa abordagem poderá ser utilizada.

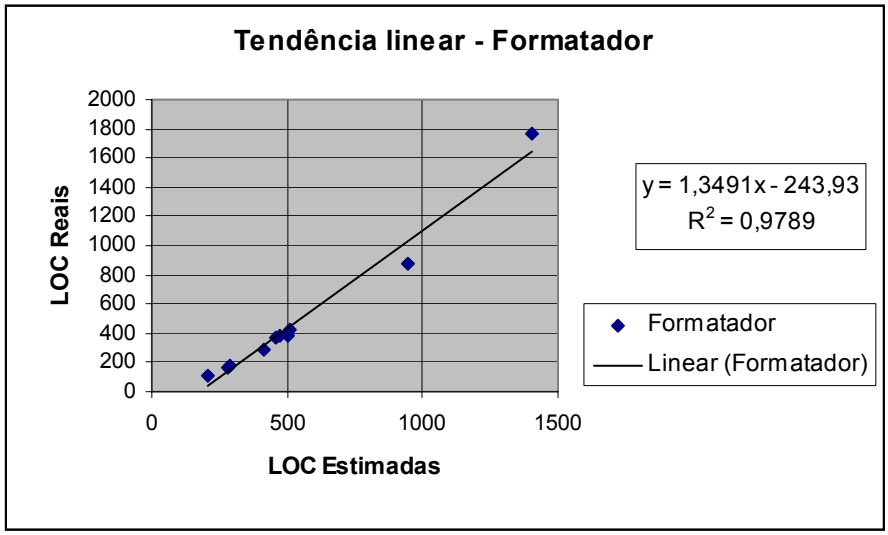

Figura 25. Tendência linear - Formatador.

A Tabela 36 apresenta os resultados para os programas da categoria Atualizador. Ao todo foram estimados e mensurados 15 programas dessa categoria. Conforme se pode observar, não há uma tendência otimista ou pessimista predominante, havendo estimativas a maior e a menor, em proporção similar. Esse pode ser um indicativo de que o método para a categoria Atualizador é mais equilibrando em relação à tendência de erro do que o método da categoria Formatador.

NÚMERO PROGRAMA	LOC ESTIMADO	LOC REAL	% ERRO
1	668	748	11,93
2	220	171	-22,22
3	390	293	-24,96
4	582	407	-30,07
5	474	512	8,06
6	860	1050	22,09
7	610	428	-29,88
8	770	954	23,85
9	837	737	-11,91
10	310	289	-6,63
11	576	627	8,80
12	310	185	-40,23
13	450	544	20,78
14	696	770	10,68
15	701	649	-7,42

Tabela 36. Resultado da mensuração - categoria Atualizador.

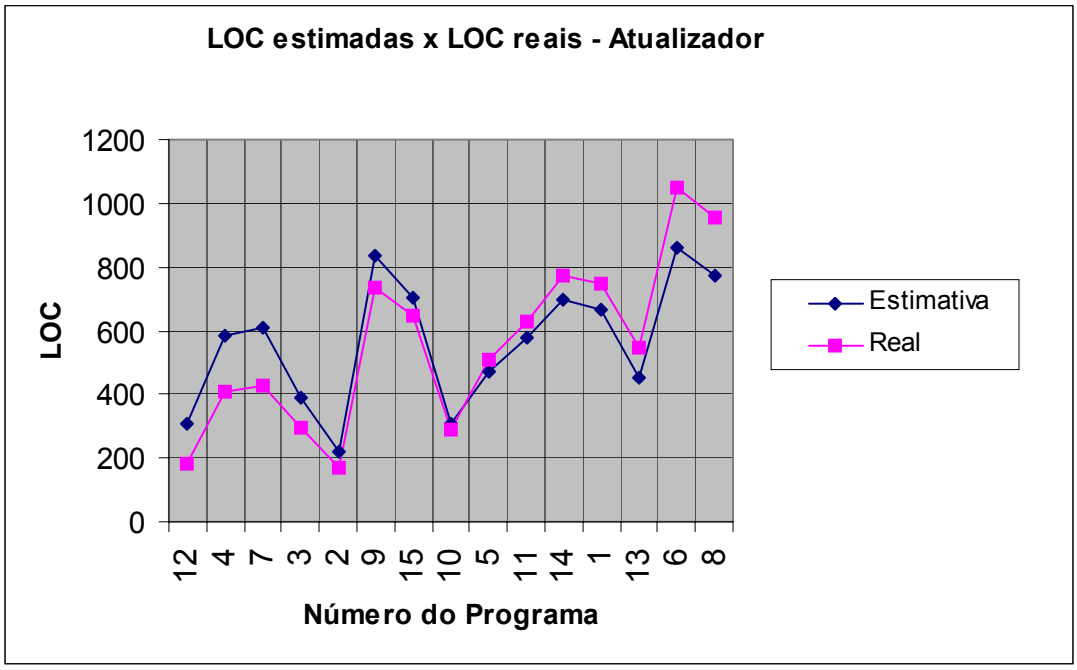

Figura 26. LOC estimadas x LOC reais - Atualizador.

Analogamente ao apresentado para a categoria Formatador, a Figura 26 apresenta, de forma gráfica, as LOC estimadas e LOC reais para os programas da categoria Atualizador. Optou-se também por classificar as informações em ordem crescente de % de erro de estimativa, para facilitar a visualização.

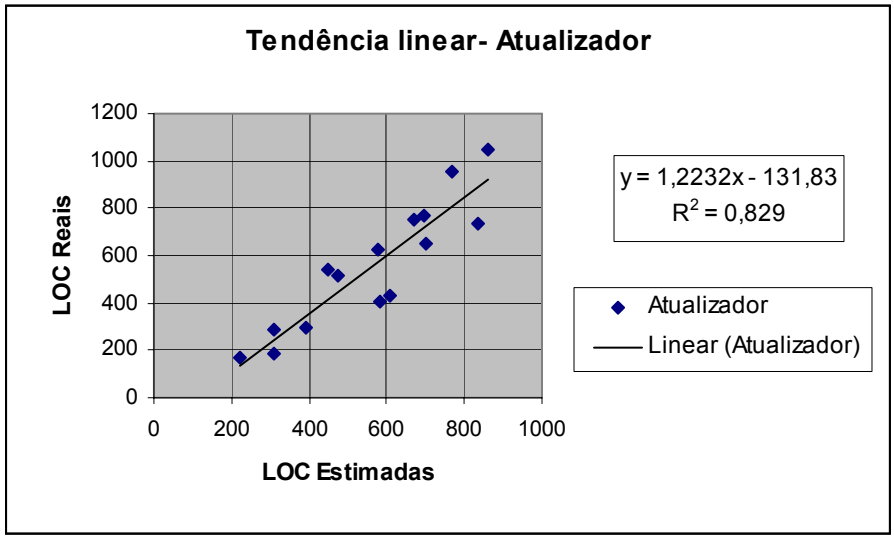

Figura 27. Tendência linear - Atualizador.

Como se pode observar, metade dos programas foi superestimada e a outra metade foi subestimada.

A Figura 27 ilustra as mesmas informações, através de um gráfico de dispersão. Nessa figura, além das informações de LOC estimadas e LOC reais para a categoria Atualizador, também aparece a tendência linear e a respectiva equação linear. O grau de correlação existente nessa categoria de programas é também bastante alto, sendo r^2 = 0,829.

6.4 Análise do estudo de caso

Os resultados do estudo de caso surpreenderam os autores desta dissertação que não imaginavam obter uma estimativa tão próxima da realidade e com um grau de correlação tão elevado. Conforme demonstrado nas seções anteriores, a instância PSPi proposta é adequada para a estimativa de programas do ambiente corporativo, utilizando a linguagem COBOL.

Buscou-se através da primeira parte do estudo de caso, montar e validar uma base inicial de tamanho relativo de LOC/*section* que pudesse auxiliar o desenvolvedor a estimar, com a maior precisão possível, o tamanho final do programa, mesmo que ele não dispusesse de uma base com seus próprios programas.

Os programas utilizados para a montagem dessa base não foram desenvolvidos por uma única pessoa, podendo conter desvios naturais de tamanho devido ao uso de diferentes estilos de programação. Não se considerou, no entanto, para o contexto deste trabalho, que isto pudesse constituir uma diferença significativa, primeiro porque a linguagem utilizada, o COBOL, não permite as inúmeras variações que outros tipos de linguagem, cuja estrutura é menos rigorosa, permitiriam (Ex.: C, C++, JAVA.). Segundo, porque as orientações propostas no guia *C36i – PROBEi Size Estimating Guidance* ajudam a reduzir possíveis variações, uma vez que levam à estimativa de cada necessidade do programa, independentemente do estilo de programação adotado.

As bases de categorias que foram desenvolvidas poderão, e deverão, ser melhoradas pelo próprio desenvolvedor, à medida que ele for incorporando a elas os seus próprios programas, auxiliando a calibrar a tendência linear. Além disso, o próprio guia com as orientações para a estimativa poderá ser melhorado à medida que o desenvolvedor analise o motivo dos desvios nas estimativas e incorpore novas regras.

A segunda etapa do estudo de caso foi conduzida com o objetivo de validar as bases de categorias montadas e as alterações propostas na instância PSPi. Nessa etapa, buscou-se o maior grau de independência possível para a realização das estimativas. Primeiramente, selecionando para executá-las uma analista de sistemas que não participou de qualquer uma das etapas anteriores deste trabalho. Segundo, selecionando para a validação uma aplicação completa, sem a exclusão de qualquer programa *batch* COBOL.

6.5 Considerações sobre o capítulo

O presente capítulo apresentou, em detalhes, como foram conduzidas as duas etapas do estudo de caso utilizado para a validação do modelo proposto. Através dos resultados, pode-se observar que as duas bases de categorias de tamanho relativo de programas apresentam uma boa correlação, podendo ser utilizadas para prever o tamanho de programas COBOL.

CAPÍTULO 7 - CONCLUSÕES E TRABALHOS FUTUROS

Conforme citado pelos autores em [REINEHR & BURNETT, 2000]:

> "O conhecimento é a matéria prima para o desenvolvimento de software, e é o engenheiro de software quem transforma conhecimento em produtos de software. Portanto, não se pode negligenciar as práticas individuais quando se deseja implementar um modelo de melhoria de processos nas organizações, seja qual for o seu porte ou nacionalidade. O PSP, se não perfeito, é um bom caminho para a indústria brasileira iniciar sua jornada rumo aos níveis internacionais de qualidade."

O que esta dissertação de mestrado propõe é uma forma de abreviar o início dessa jornada, oferecendo adicionais valiosos para complementar o modelo original PSP, visando adaptá-lo às necessidades do ambiente corporativo legado. As extensões propostas através da instância PSPi auxiliam a empresa a promover melhorias em um dos aspectos mais críticos do planejamento de projetos de software: a determinação do tamanho da atividade.

Este capítulo apresenta as conclusões obtidas por esta dissertação e os trabalhos que dela podem ser derivados.

7.1 Conclusões

A primeira contribuição oferecida por esta dissertação é a visão geral e abrangente dos modelos e normas que envolvem a definição e a melhoria dos processos de software, apresentada no capítulo 2. Como se pode nele observar, muitas dessas versões acabaram de ser publicadas ou estão para ser publicadas nos próximos meses. Isso demonstra que é um segmento que se encontra em plena evolução, como toda a disciplina de engenharia de software. Através dessa visão geral é possível para o leitor buscar as referências necessárias e atualizadas para aprofundar seu conhecimento nos modelos e/ou normas de interesse da organização.

A segunda contribuição, diz respeito ao detalhamento do próprio modelo PSP e sua situação de utilização no Brasil e nos outros países, apresentada no capítulo 3. Devido ao pouco, ou quase nenhum, conhecimento do PSP no Brasil, o próprio

aprofundamento no conhecimento dos detalhes de funcionamento dos processos, formulários e roteiros do modelo já constitui, por si só, uma grande contribuição desta dissertação. Além do detalhamento do modelo original, este capítulo ainda oferece ao leitor uma análise dos pontos importantes para o desenvolvimento de software, os cinco "P"s, e como eles se relacionam com o PSP. Esta análise crítica possibilita que o leitor compreenda em que pontos o PSP poderá ajudá-lo e quais são os pontos em que o modelo deixa margem a complementações.

A terceira, e mais importante contribuição desta dissertação, é a instância proposta PSPi. Essa instância, através de suas extensões ao modelo PSP, oferece uma forma de auxiliar o desenvolvedor na tarefa de estimar o tamanho do programa que ele vai desenvolver. As bases de categorias de tamanho relativo de *proxies*, apresentadas separadamente para os programas do tipo Formatador e Atualizador, comprovaram ser úteis no planejamento do tamanho quando combinadas com as orientações para a concepção do programa, descritas na extensão *C36i – PROBEi Size Estimating Guidance.*

A quarta contribuição é a ferramenta PSPPlus-i, desenvolvida utilizando interface gráfica e amigável, que apóia a execução dos processos propostos na instância PSPi. Devido ao seu desenvolvimento modularizado, é possível facilmente extendê-la para contemplar novas funcionalidades.

Ao optar por trabalhar todas as extensões em inglês, idioma original do PSP, esta dissertação mantém a compatibilidade com o modelo original e evita que o usuário sinta-se confuso e perdido entre a nomenclatura dos diversos formulários originais e dos formulários propostos na instância.

7.2 Trabalhos futuros

Esta dissertação de mestrado oferece uma visão abrangente e atual, porém não completamente exaustiva, da questão da qualidade dos processos de software no país e no mundo. Além disto, seleciona e adapta um dos modelos que vem sendo internacionalmente utilizado, o PSP, para ambientes corporativos de grande porte. Sobre esta adaptação, realiza um estudo de caso que comprova a aplicabilidade e a efetividade do método e das bases propostas.

Assim como a engenharia de software encontra-se em plena evolução, caminhando rumo à sua maturidade, também os modelos e as normas de processos de software vêm sendo construídos, criticados e melhorados. Importantes contribuições são feitas por esta dissertação ao modelo individual de melhoria de processos PSP, no entanto, ainda há o que evoluir e contribuir para o amadurecimento desta e de outras práticas da engenharia de software, como:

- A incorporação da instância PSPi ao modelo TSP [HUMPHREY, 1999], de modo que possa ser o ponto de partida para as estimativas dos projetos de software em equipes maiores do ambiente corporativo legado;
- A extensão da instância PSPi para o tratamento dos programas do tipo Receptor, utilizando diferentes linguagens de quarta geração de mercado, incluindo: a definição de um padrão para a codificação nessas linguagens, definição de um padrão para a contagem de LOC, elaboração de orientações para a estimativa e montagem da base de categorias de tamanho relativo;
- A incorporação no PSPi de alterações que passem a tratar as fases iniciais do ciclo de vida, especificamente a elicitação de requisitos, não tratada extensivamente nem pelo PSP, nem pelo PSPi, usando, por exemplo, a abordagem de eventos de negócio combinada com análise por pontos de função, proposta em [WILEY, 1999];
- A adaptação do PSPi para tornar-se mais aderente aos conceitos do CMMI [SEI, 2000a][SEI, 2000b], uma vez que sua base foi o PSP original que, por sua vez, derivou do SW-CMM [PAULK et alli, 1994];
- A extensão da ferramenta PSPPlus-i para incorporar as análises estatísticas necessárias para a montagem da base de categorias de tamanho relativo, atualmente efetuadas externamente à ferramenta;
- A generalização da ferramenta PSPPlus-i, com a definição das regras de cada linguagem de programação em banco de dados, tornando uma nova instanciação mais fácil e automatizada;
- A integração da ferramenta PSPPlus-i com ferramentas CASE e de gerenciamento de projetos, tornando-a mais robusta e aderente ao ambiente de desenvolvimento como um todo;

- A análise da correlação das demais informações mensuradas nos programas da base de categorias (quantidade de arquivos, quantidade de *copys* etc.) com a estimativa de tamanho de programa, como forma de refinar o guia de estimativas proposto.

E, finalmente, àqueles que se dedicam a dar continuidade e evoluir as idéias de seus pares, produzindo arte e ciência, uma citação de Oswald de Andrade:

> "Não fique preso às tradições. Adicione elementos simples, e assim encontrará a arte."

REFERÊNCIAS BIBLIOGRÁFICAS

[ABNT, 1996] ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 13596 – Tecnologia de informação - Avaliação de produto de *software* – Características de qualidade e diretrizes para o seu uso** (Versão brasileira da norma ISO/IEC 9126, 1991). Rio de Janeiro: ABNT, 1996, 10 p.

[ABNT, 1997] ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO 9000-3:1997 – Normas de Gestão da Qualidade e Garantia da Qualidade – parte 3, Diretrizes para Aplicação da NBR 9001 ao desenvolvimento, fornecimento e manutenção de *software.*** Rio de Janeiro: ABNT, 1997.

[ABNT, 1998] ABNT – ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 12207 – Tecnologia de informação - Processos de ciclo de vida de *software*.** Rio de Janeiro: ABNT, 1998, 35 p.

[ALBUQUERQUE & MEIRA, 1997] ALBUQUERQUE, JONES DE OLIVEIRA; MEIRA, SÍLVIO R. **PSP-JOA Processo de Software Pessoal – Uma abordagem Orientada a JAVA.** Dissertação de Mestrado. Recife: Universidade Federal de Pernambuco, 1997, 141p.

[ALBUQUERQUE et alli, 1998] ALBUQUERQUE, JONES DE OLIVEIRA; MEIRA, SÍLVIO R.; COELHO, CLAUDIONOR **Practical Limits of the PSP Model.** In proceedings of SoSt'98 – Symposium on Software Technology, Buenos Aires, Argentina, pp. 95-100, 1998.

[BATE et alli, 1995] BATE, ROGER; KUHN, DOROTHY; WELLS, CURT; ARMITAGE, JAMES; CLARK, GLORIA; CUSICK, KERINIA; GARCIA, SUZANNE; HANNA, MARK; JONES, ROBERT; MALPASS, PETER; MINNICH, ILENE; PIERSON, HAL; POWEL, TIM; REICHNER, AL .**A Systems Engineering Capability Maturity Model - Version 1.1 (CMU/SEI-95-MM-003)**. Pittsburgh, PA:SEI, Carnegie Mellon University, 1995.
http://www.sei.cmu.edu/publications/documents/95.reports/95.mm.003.html, 01/08/1999.

[BROOKS, 1987] BROOKS JR, FREDERICK P. **No Silver Bullet: Essence and Accidents of Software Engineering**. IEEE Computer, v.20, n.4, p9-10, 1987.

[BROOKS, 1995] BROOKS JR, FREDERICK P. **The Mythical Man-Month – Essays on Software Engineering - Anniversary Edition.** 13ª. edição. Reading, MA: Addison-Wesley Publishing Company, 1995, 322 p.

[CANADA, 1992] CANADA, BELL. Trillium: Telecom Software Product Development Process Capability Assessment Model. Draft 2.2. Bell Canada, 1992.

[COOPER et alli, 1999] COOPER, JACK; FISHER, MATTHEW; SHERER, S.WAYNE. **Software Aquisition Capability Maturity Model (SA-CMM) – Version 1.02 (CMU/SEI-99-TR-002)**. Pittsburgh, PA:SEI, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/publications/documents/99.reports/99tr002, em 01/08/1999.

[CURTIS et alli, 1988] CURTIS, BILL; KRAESNER, H.; ISCOE, N. **A Field Study of the Software Design Process for Large Systems.** Communications of the ACM 31, 11, pp. 1268-1287, 1988.

[CURTIS et alli, 1995a] CURTIS, BILL; HEFLEY, WILLIAM; MILLER, SALLY. **Overview of the People Capability Maturity Model (CMU/SEI-95-MM-01).** Pittsburgh, PA: SEI, Carnegie Mellon University, 1995.
http://www.sei.cmu.edu/pubs/documents/95.reports/pdf/mm01.95.pdf, em 30/08/1998.

[CURTIS et alli, 1995b] CURTIS, BILL; HEFLEY, WILLIAM; MILLER, SALLY. **People Capability Maturity Model (CMU/SEI-95-MM-02)**. Pittsburgh, PA: SEI, Carnegie Mellon University, 1995.
http://www.sei.cmu.edu/pubs/documents/95.reports/pdf/mm02.95.pdf, em 23/02/1999.

[CURTIS et alli, 2001] CURTIS, BILL; HEFLEY, WILLIAM; MILLER, SALLY. **Overview of the People Capability Maturity Model (CMU/SEI-01-MM-001).** Pittsburgh, PA: SEI, Carnegie Mellon University, 2001.
http://www.sei.cmu.edu/publications/documents/01.reports/01mm001.html, em 17/09/01.

[CSE, 1998] CSE – Centre for Software Engineering. **Software Process Improvement – Case Study.** Dublin: Spire Book, Número 5, November 1998.
ftp://ftp.cse.dcu.ie/pub/spire/advent.pdf, em 08/11/1999.

[DELLIEN, 1997] DELLIEN, OLOF. **The Personal Software Process in Industry.** Thesis of Master Science. Lund: Lund Institute of Technology – Department of Communication Systems, 1997, 41p.

[DISNEY & JOHNSON, 1998] DISNEY, ANNE. **Data Quality Problems in the Personal Software Process**. Thesis of Master of Science in Information and Computer Science. Honolulu: University of Hawaii, 1998, 95p.
http://csdl.ics.hawaii.edu/techreports/98-08/98-08.pdf, em 13/09/1999.

[ESCALA & MORISIO, 1998] ESCALA, DAVID; MORISIO, MAURIZIO. **A metric suite for a team PSP.** In proceedings of The Fifth International Software Metrics Symposium, Bethesda, Maryland, pp. 61-71, 1998.

[FERGUSON et alli, 1997] FERGUSON, PAT; HUMPHREY, WATTS S.; KHAJENOORI, SOHEIL; MACKE, SUSAN; MATVYA; ANNETTE. **Introducing the Personal Software Process: Three industrial cases.** IEEE Computer, Volume 30, Número 5, pp. 24-31, Maio 1998.

[FERGUSON et alli, 1999] FERGUSON, PAT; LEMAN, GLORIA; PERINI, PRASAD; RENNER, SUSAN; SESHAGIRI, GIRISH. **Software Process Improvment Works! Advanced Information Services Inc.** (CMU/SEI-99-TR-027). Pittsburgh,PA: SEI, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr027.pdf, em 23/11/1999.

[FLORAC & CARLETON, 1999] FLORAC, WILLIAM A.; CARLETON, ANITA D. **Measuring the Software Process – Statistical Process Control for Software Process Improvement.** Reading,MA: Addison-Wesley Publishing Company, 1999, 250 p.

[GIBSON, 1997] GIBSON, RICK. **Applied Software Process Improvement**. In proceedings of the Americas Conference on Information Systems, pp. 596-598, 1997.
http://hsb.baylor.edu/ramsower/ais.ac.97/gibson.html, em 14/02/2000.

[HAYES & OVER, 1997] HAYES, WILL & OVER, JAMES. **The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers (CMU/SEI-97-TR-001).** Pittsburgh,PA: SEI, Carnegie Mellon University, 1997.
http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97tr001.pdf, em 31/08/1998.

[HAYES, 1998] HAYES, WILL. **Using a Personal Software Process to Improve Performance.** In proceedings of The Fifth International Software Metrics Symposium, Bethesda, Maryland, pp. 61-71, 1998.

[HUMPHREY, 1994] HUMPHREY, WATTS S. **The Personal Software Process**. Software Process Newsletter, Technical Council on Software Engineering, Volume 13, Número 1, Sept 1994, pp SPN 1-3.
http://www.sei.cmu.edu/publications/articles/psp.pdf, em 31/08/1998.

[HUMPHREY, 1995] HUMPHREY, WATTS S. **A Discipline for Software Engineering.** Reading, MA: Addison-Wesley Publishing Company, 1995, 789 p.

[HUMPHREY, 1997] HUMPHREY, WATTS S. **Introduction to the Personal Software Process.** Reading,MA: Addison-Wesley Publishing Company, 1997, 278p.

[HUMPHREY, 1998] HUMPHREY, WATTS S. **Disciplined Software Teams.** In proceedings of IX CITS Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba-PR, pp. 3-31, 1998.

[HUMPHREY, 1999] HUMPHREY, WATTS S. **Introduction to the Team Software Process.** Reading, MA: Addison-Wesley Publishing Company, 1999, 463 p.

[HUMPHREY, 2000a] HUMPHREY, WATTS S. **The Personal Software Process (CMU/SEI-2000-TR-022).** Pittsburghh, PA: SEI, Carnegie Mellon University, 2000.
http://www.sei.cmu.edu/publications/documents/00.reports/00tr022.html, em 01/10/2001.

[HUMPHREY, 2000b] HUMPHREY, WATTS S. **The Team Software Process (CMU/SEI-2000-TR-023).** Pittsburghh, PA: SEI, Carnegie Mellon University, 2000.
http://www.sei.cmu.edu/publications/documents/00.reports/00tr023.html, em 01/10/2001

[ISO/IEC, 1991] ISO/IEC – INTERNATIONAL ORGANIZATION FOR STANDARDIZATION/ INTERNATIONAL ELECTROTECHNICAL COMISSION. **ISO/IEC 9126 – Information technology - Software product evaluation – Quality characteristics and guidelines for their use**. ISO/IEC, 1991, 13 p.

[ISO/IEC, 1998a] ISO/IEC - INTERNATIONAL ORGANIZATION FOR STANDARDIZATION/ INTERNATIONAL ELECTROTECHNICAL COMISSION. **ISO/IEC 14598: part 5 – Information technology – Software Product Evaluation – Process for Evaluators**. ISO/IEC, 1998.

[ISO/IEC, 1998b] ISO/IEC – INTERNATIONAL ORGANIZATION FOR STANDARDIZATION/ INTERNATIONAL ELECTROTECHNICAL

COMISSION. **ISO/IEC TR 15504: parts 1-9 – Information technology - Software Process Assessment**. ISO/IEC, 1998.

[ISO/IEC, 1998c] ISO/IEC – INTERNATIONAL ORGANIZATION FOR STANDARDIZATION/ INTERNATIONAL ELECTROTECHNICAL COMISSION. **ISO/IEC TR 15271:1998 – Information technology – Guide for ISO/IEC 12207 (Software Lifecycle Process)**. ISO/IEC, 1998.

[ISO/IEC, 1999] ISO/IEC - INTERNATIONAL ORGANIZATION FOR STANDARDIZATION/ INTERNATIONAL ELECTROTECHNICAL COMISSION. **ISO/IEC 14598: parts 1,2,3,4 e 6 – Information technology – Software Product Evaluation**. ISO/IEC, 1999.

[JACOBSON et alli, 1999] JACOBSON, IVAR; BOOCH, GRADY; RUMBAUGH, JAMES. **The Unified Software Development Process.** Reading, MA: Addison-Wesley Publishing Company, 1999, 463 p.

[KRUCHTEN, 2000] KRUCHTEN, PHILIPPE. **The Rational Unified Process – An Introduction – Second Edition.** Reading, MA: Addison-Wesley Publishing Company, 2000, 298 p.

[MACHADO et alli, 2000] MACHADO, CRISTINA; REINEHR, SHEILA S.; CALSAVARA, ALCIDES; BURNETT, ROBERT C. **Aderência do RUP à Norma NBR ISO/IEC 12207.** Anais do II SIMPROS – Simpósio Brasileiro de Melhoria de Processo de Software, São Paulo-SP, pp. 75-86, 2000.

[MAIDANTCHIK et alli, 1999] MAIDANTCHIK, CARMEM; ROCHA, ANA REGINA; XEXÉO, GERALDO. **Software Process Standardization for Distributed Working Groups.** In proceedings of The 4th IEEE International Software Engineering Standards Symposium and Forum (ISESS'99), Curitiba-PR, pp. 153-156, 1999.

[MC ANDREWS, 2000] MC ANDREWS, DONALD R. **The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices (CMU/SEI-2000-TR-015).** Pittsburghh, PA: SEI, Carnegie Mellon University, 2000.
http://www.sei.cmu.edu/publications/documents/00.reports/00tr015.html, em 01/10/2001

[MCMENAMIN & PALMER, 1984] MCMENAMIN, STEVE & PALMER, JOHN. **Essential Systems Analysis.** New York: Yourdon Press, 1984, páginas

[MCT, 2000] MCT – MINISTÉRIO DA CIÊNCIA E TECNOLOGIA. **Qualidade no Setor de Software Brasileiro – 1999.** Brasília: Ministério da Ciência e Tecnologia, 2000.
http://www.mct.gov.br/Temas/info/dsi/palestra/palestras.htm

[MORISIO, 1999] MORISIO, MAURIZIO. **Applying the PSP in industry.** Experience Report. Maryland: University of Maryland, 1999.

[PAULK et alli, 1993] PAULK, MARK; CURTIS, BILL; CHRISSIS, MARY BETH; WEBER, CHARLES. **Capability Maturity Model for Software - Version 1.1 (CMU/SEI-93-TR-024)**. Pittsburgh, PA:SEI, Carnegie Mellon University, 1993.
http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html, 01/08/1999.

[PAULK et alli, 1994] PAULK, MARK; CURTIS, BILL; CHRISSIS, MARY BETH; WEBER, CHARLES. **Capability Maturity Model for Software: Guidelines for Improving the Software Process**. Reading,MA: Addison-Wesley Publishing Company, 1994, 441 p.

[PMI, 2000] PROJECT MANAGEMENT INSTITUTE. **A Guide to the Project Management Body of Knowledge – PMBOK Guide.** PA: Project Management Institute, 2000, 216 p.

[PRECHELT & UNGER, 2001] PRECHELT, LUTZ; UNGER, BARBARA. **An Experiment Measuring the Effects of Personal Software Process (PSP) Training.** IEEE Transactions on Software Engineering, 27(5), pp. 465-472, 2001.

[PRESSMAN, 1997] PRESSMAN, ROGER S. **Software Engineering – A Practitioner's Approach.** New York: The McGraw-Hill Companies, 1997, 852 p., 4ª ed.

[REINEHR & BURNETT, 2000] REINEHR, SHEILA S.; BURNETT, ROBERT C. **PSP: uma boa opção para a indústria brasileira?** Anais da XI CITS Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba-PR, pp. 85-96, 2000.

[REINEHR, 2001] REINEHR, SHEILA S. **Utilização do PSP no desenvolvimento de software.** São Leopoldo, RS: Gráfica da UNISINOS, 2001, 100p.

[ROBERTSON & ROBERTSON, 1999] ROBERTSON, SUZANNE & ROBERTSON, JAMES. **Mastering the Requirements Process.** Harlow: ACM Press & Addison-Wesley Publishing Company, 1999, 404 p.

[ROCHA & MACHADO, 2000] ROCHA, ANA R.; MACHADO, LUIS FILIPE C. **Minicurso 4 - Uso de Normas e Modelos de Maturidade na Definição de Processos de Software.** XI CITS Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba-PR, 2000, 22 p.

[ROCHA, MALDONADO & WEBER, 2001] ROCHA, ANA R.; MALDONADO, JOSÉ C. **Qualidade de Software – Teoria e Prática.** São Paulo: Pearson Education do Brasil/Makron Books LTDA., 2001, 303 p.

[SEACORD et alli, 2001] SEACORD, ROBERT; DORDA-COMELLA, SANTIAGO; LEWIS, GRACE; PLACE, PAT; PLAKOSH, DAN. **Legacy System Modernization Strategies (CMU/SEI-2001-TR-025).** Pittsburgh, PA: SEI, Carnegie Mellon University, 2001.
http://www.sei.cmu.edu/publications/documents/01.reports/01tr025.html, em 14/08/2001.

[SEI, 1999a] SOFTWARE ENGINEERING INSTITUTE - SEI. **A Specification for the CMMI Product Suite, Version 1.4**. Pittsburgh, PA:SEI, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/cmm/cmmi/specs/aspec1.4.html, em 02/08/1999.

[SEI, 1999b] SOFTWARE ENGINEERING INSTITUTE - SEI. **Capability Maturity Model – Integrated – Systems/Software Engineering – CMMI-SE/SW – Version 0.2b, Continuos Representation, Volume I e II**. Pittsburgh, PA:SEI, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/cmm/cmmi/public-review/public-review.html, em 23/11/1999.

[SEI, 1999c] SOFTWARE ENGINEERING INSTITUTE - SEI. **Capability Maturity Model – Integrated – Systems/Software Engineering – CMMI-SE/SW – Version 0.2b, Staged Representation, Volume I e II**. Pittsburgh, PA:SEI, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/cmm/cmmi/public-review/public-review.html, em 23/11/1999.

[SEI, 2000a] SOFTWARE ENGINEERING INSTITUTE - SEI. **CMMI for Systems Engineering/Software Engineering, Version 1.02 (CMMI-SE/SW, V1.02), Staged Representation (CMU/SEI-2000-TR-018).** Pittsburgh, PA:SEI, Carnegie Mellon University, 2000.
http://www.sei.cmu.edu/publications/documents/00.reports/00tr018.html, em 17/09/2001.

[SEI, 2000b] SOFTWARE ENGINEERING INSTITUTE - SEI. **CMMI for Systems Engineering/Software Engineering, Version 1.02 (CMMI-SE/SW, V1.02),**

Continuous Representation (CMU/SEI-2000-TR-019). Pittsburgh, PA:SEI, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tr019.html, em 17/09/2001.

[SEI, 2000c] SOFTWARE ENGINEERING INSTITUTE - SEI. **CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development, Version 1.02 (CMMI-SE/SW/IPPD, V1.02), Continuous Representation (CMU/SEI-2000-TR-031).** Pittsburgh, PA:SEI, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tr031.html, 17/09/2001.

[SEI, 2000d] SOFTWARE ENGINEERING INSTITUTE - SEI. **CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development, Version 1.02 (CMMI-SE/SW/IPPD, V1.02), Staged Representation (CMU/SEI-2000-TR-030).** Pittsburgh, PA:SEI, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tr030.html, em 17/09/2001.

[SILVA & MASIERO, 1998] SILVA, DJALMA D. & MASIERO, PAULO C. **Uma Proposta de Versão e Ferramenta do PSP para o Domínio de Aplicações Comerciais.** Anais da IX CITS Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba-PR, pp. 157-168, 1998.

[SILVA & SPÍNOLA, 1999] SILVA, MARCOS ANTONIO RODRIGUES & SPÍNOLA, MAURO DE MESQUITA**. Um método para a utilização do modelo PSP (Personal Software Process).** Dissertação de Mestrado. São Paulo: Universidade Paulista, 1999.

[SOMMERVILLE & SAWYER, 1997] SOMMERVILLE, IAN & SAWYER, PETE. **Requirements Engineering – A good practice guide.** Chichester: John Wiley & Sons, 1997, 391 p.

[SPIEGEL, 1977] SPIEGEL, MURRAY. **Probabilidade e Estatística.** Rio de Janeiro: Makron Books do Brasil Editora LTDA, 1977, 518p.

[WENCESLAU et alli, 2001] WENCESLAU, LUCIANA; DAKKACHE, MARIANA; REINEHR, SHEILA S.; BURNETT, ROBERT B. **PSP2000 – Uma Ferramenta para automatizar o PSP.** Anais da XII CITS Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba-PR, pp. 141-152, 2001.

[WILEY, 1999] WILEY, BILL. **Essential Systems Requirements – A practical Guide to Event-Driven Mehods.** Reading,MA: Addison-Wesley Publishing Company, 1999, 251p.

[ZANLORENCI & BURNETT, 1999] ZANLORENCI, EDNA P. & BURNETT, ROBERT C. **Descrição e qualificação de requisitos – um modelo aplicável à análise e validação**. Dissertação de Mestrado. Curitiba: Pontifícia Universidade Católica do Paraná, 1999.