UNIVERSIDADE FEDERAL DE PELOTAS
INSTITUTO DE FÍSICA E MATEMÁTICA
Curso de Bacharelado em Informática

Uma Disciplina Para a Engenharia de *Software*: Estudo do *Personal Software Process* (PSP), Proposto Por Watts Humphrey (A Profissionalização do Desenvolvedor de *Software*)

Por

José Wilson da Silva Júnior

Projeto de Conclusão de Curso apresentado ao Curso de Bacharelado de Informática, Instituto de Física e Matemática, Universidade Federal de Pelotas.
Orientadora: Profa. Flávia Azambuja Carvalho, MSc.

PELOTAS
Dezembro de 2000

Agradecimento

"Se não morre aquele que escreve um livro ou planta
uma árvore, com mais razão não morre o educador,
que semeia vida e escreve na alma".
Autor desconhecido

Este trabalho foi **inspirado** por conversas com a Professora Flávia Azambuja, durante as aulas de Análise e Projeto de Sistemas, no sétimo semestre do curso de informática (primeiro semestre de 2000).

Posteriormente, ao apresentar a proposta do trabalho de conclusão de curso, a Flávia **incentivou** a continuidade do trabalho.

Durante todo o período de pesquisas e composição, a Flávia colocou-se sempre a disposição, **apoiando** e **estimulando**.

Quero manifestar meus sinceros agradecimentos à Flávia, profissional que faz juz à palavra **Professora**.

José Wilson da Silva Júnior
Janeiro de 2001

Sumário

Lista de Abreviaturas e Siglas

- PSP[1] (*Personal Software Process*): Processo de *Software* Pessoal.
- SEI (*Software Engineering Institute*): Instituto de Engenharia de *Software*.
- TSP (*Team Software Process*): Processo de *Software* de Time (Grupo).
- COQ *(Cost of Quality*): Custo de Qualidade.
- CR (*Code Review*): Revisão de Código.
- CMM (*Capability Maturity Model*): Modelo de Maturidade de Capacidade.
- DR (*Design Review*): Revisão de Projeto.
- DRL (*Defect Removal Leverage*): Média de Remoção de Defeito.
- IPD (*Institute for Program Structure and Data Organization*): Instituto para Estruturação de Programas e Organização de Dados.
- KLOC (*One thousand lines of code*): mil linhas de código.
- LOC (*Lines of Code*): Linhas de Código.
- PROBE (*Proxy Based Estimating*): Estimativa Baseada em *Proxy*.
- PIP (*Process Improvement Proposal*): Proposta de Melhoria de Processo.
- Pg.: página.
- KPA (*Key Process Area*): Área Chave de Processo.
- SGBD: Sistema Gerenciador de Banco de Dados.
- ESI (*European Software Institute*): Instituto Europeu de *Software*
- ESSI (*European Systems & Software Initiative*): Iniciativa Européia de Sistemas e *Software*.

[1] PSP, Processo de *Software* Pessoal, TSP, Processo de *Software* de Time, CMM, Modelo de Maturidade de Capacidade são marcas registradas da Carnegie Mellon University.

Lista de Figuras:

Lista de Tabelas

Resumo

A qualidade é um dos assuntos mais importantes na indústria de *software* hoje e provavelmente será ainda mais importante no século 21. As aplicações de *software* aumentam em tamanho em uma ordem de magnitude a cada 5-10 anos, e assim ocorre também com o número de *bugs*. Numerosos métodos surgiram para resolver o problema de defeitos de *software*. O PSP (Processo de *Software* Pessoal) [HUM 95], um processo disciplinado de auto-melhoria, é um dos mais proeminentes. O Processo de *Software* Pessoal é um processo de desenvolvimento de *software* que permite ao indivíduo aplicar a disciplina de nível industrial na sua prática pessoal. O PSP provê aos engenheiros de *software* um modo para melhorar a qualidade, a predictibilidade e a produtividade do seu trabalho. É projetado para resolver as necessidades de melhoria de engenheiros individuais e de organizações de *software* pequenas. O livro de Humphrey [HUM 95] provê uma sucessão definida de passos de melhoria de processo, junto com uma avaliação de desempenho em cada passo. Este texto examina primeiramente a situação atual do desenvolvimento de *software*. A seguir apresenta uma descrição sumária de PSP, identificando as práticas do Processo de *Software* Pessoal. Finalmente, apresenta os resultados da aplicação do PSP e outros assuntos relacionados. O texto relata brevemente os resultados de um estudo importante para todos aqueles que administram ou desenvolvem *software*. O estudo examina o impacto do Processo de *Software* Pessoal no desempenho de 298 engenheiros de *software*. [HAY 97] O presente trabalho conclui que a filosofia de PSP é pertinente para o desenvolvedor de *software* e pode melhorar a qualidade de *software* por ele produzido. Porém, modos de reduzir os custos da implementação do PSP são necessários antes do método ser adotado amplamente. O autor do presente trabalho acredita que o custo para executar o PSP pode ser reduzido provendo apoio automatizado para as atividades do método. Também acredita que o PSP pode se tornar uma base para desenvolver um processo de *software* pessoal customizado.

Palavras-chave: qualidade, defeito, processo, engenharia de *software*, Processo Pessoal de *Software*, método, projeto.

Abstract

Quality is one of the most important issues in software industry today and it will likely be even more important in the 21st century. Software applications increase in size with an order of magnitude every 5-10 years, and so does the number of "bugs". Numerous methods have arisen to address the problem of software defects. The PSP (Personal Software Process) [HUM 95], a disciplined self-improvement process, is one of the most prominent. The Personal Software Process is a software development process that allows the individual to apply industrial level discipline to his or her practice. The PSP provides software engineers a way to improve the quality, predictability, and productivity of their work. It is designed to address the improvement needs of individual engineers and small software organizations. Humphrey's textbook [HUM 95] provides a defined sequence of process improvement steps coupled with performance feedback at each step. This text firstly examines the current situation of the software development. To proceed it presents a summary des*cript*ion of PSP, identifying the beginnings and practices of the Personal Software Process. Finally, it presents the results of the application of PSP and other related subjects. This text also resumes briefly the results of a study that is important to everyone who manages or develops software. The study examines the impact of the Personal Software Process on the performance of 298 software engineers. [HAY 97] This text concludes that the philosophy of PSP is relevant for software developers, and can improve software quality. However, ways of reducing the cost of implementing PSP are necessary before the method is likely to be widely adopted. It is believed that the cost to perform PSP can be reduced through providing automated support for PSP activities. It is also believed that the PSP can become a basis for developing a custom personal software process.

Key-words: quality, defect, process, software engineering, Personal Software Process, method, project.

1 Introdução

> "Se os engenheiros construíssem prédios como os programadores escrevem programas, um único pica-pau seria capaz de destruir a civilização".
> Segunda Lei de Weinberg, extraído de [JUN 00]

1.1 Introdução

O desenvolvimento de *software*, hoje, está mais próximo de uma arte do que de uma ciência. Após um período de aprendizado bastante genérico nas universidades, os profissionais da área desenvolvem seus próprios métodos e técnicas; alguns desses profissionais desenvolvem produtos de boa qualidade, outros nem tanto.[HUM 95] Não existe, nas universidades, uma aprendizagem de métodos formais para o desenvolvimento de *software* – pelo menos ao nível individual.

A diferença entre profissionais de informática formados em instituições de nível superior e aqueles que adquirem a sua formação em cursos de informática para leigos não é tão nítida como deveria ser.

Como resultado dessa situação, as empresas, normalmente, valorizam mais o desenvolvedor de fato do que aquele que apenas possui um diploma universitário. É comum vermos profissionais não qualificados à nível acadêmico ocupando diversos cargos nos setores de informática das empresas.

Entretanto, comparando essa situação com a de outras profissões, constata-se que a situação é completamente diferente: nenhum empresário ou dirigente contrataria, por exemplo, um arquiteto ou médico formado em um curso que não de nível superior. Isto seria inconcebível. Tal situação ocorre devido a uma lacuna existente na formação dos desenvolvedores, nas universidades, e não por causa dos inúmeros cursos de treinamento em informática existentes ou pelo brilhantismo de pessoas autodidatas.

Assim, é necessário que os profissionais oriundos das universidades possuam métodos, técnicas e ferramentas cientificamente elaboradas, que diferenciem o seu trabalho do trabalho dos profissionais leigos (aqueles que não têm uma formação acadêmica na área).

Outro problema diz respeito à crescente complexidade dos novos projetos de *software*. Existe, cada vez mais, a necessidade de programas melhores, maiores, mais

rápidos e mais baratos. A prática atual não tem correspondido à essas expectativas. [HUM 95] Nas palavras de Pressman "*... a set of software-related problems has persisted throughout the evolution of computer-based systems, and these problems continue to intensify.*" [PRE 97] O desenvolvimento intuitivo de *software* deve ceder lugar ao desenvolvimento disciplinado, metódico e exato - enfim, científico.

A importância deste assunto fica mais óbvia ao se estudar sistemas críticos, como plantas nucleares ou dispositivos de cuidados médicos que são computadorizados cada vez mais todos os anos. Em uma apresentação[2], Humphrey é bem enfático com relação a essa questão:

"*E se sua vida dependesse de software?*

- *O mundo está mudando*
- *O software pilota os nossos aviões agora*
- *O software acelera, guia e freia nossos automóveis*
- *O software transfere nosso dinheiro*
- *O software está presente em quase todos os aspectos de nossas vidas*

É só uma questão de tempo:

- *Até mesmo o software extensamente usado tem muitos defeitos*
- *O software agora invade nossas vidas de forma que estes defeitos inevitavelmente causarão problemas*
- *Embora existam poucos casos até esta data, programas defeituosos ou difíceis de usar vão, quase certamente:*
  - *matar ou mutilar pessoas*
  - *romper negócios*
  - *provocar desastres ou acidentes devastadores"*

Estudiosos da Engenharia de *Software* vêm pesquisando todas essas questões e já propuseram alguns modelos para resolver os problemas, como:

- CMM – *Capability Model Maturity*
- PSP – *Personal Software Process*
- SPICE – *Software Process Improvement and Capability Determination*
- P-CMM - *People Capability Maturity Model*
- ISO 9000-3 - Normas para aplicação da série ISO 9000 em processos de *software*
- ISO 12207 - Processos do Ciclo de Vida do *Software*
- Modelo Trillium
- Metodologia Bootstrap
- Engenharia de *Software Cleanroom*

[2] "What if Your Life Depended on Software" , apresentação no Power Point, da Microsoft, obtida no site http://davidfrico.com/psp-online-f.htm, de David F. Rico, consultor internacional de alta tecnologia.

Este trabalho é focado no *Personal Software Process* – PSP, proposto por Watts Humphrey, em 1995, no seu livro *A Discipline for Software Engineering*, por ser a base sobre a qual podem ser edificados outros métodos de nível mais alto, isto é, no nível organizacional.[3]

1.2 Objetivos

O objetivo primordial do presente trabalho é o estudo, a discussão e a sistematização do método proposto por Watts Humphrey em sua obra, o qual tem, por sua vez, o objetivo de auxiliar a cada Engenheiro de *Software*, individualmente, a criar o seu próprio processo pessoal de *software* (PSP) e a melhorar a qualidade de seus produtos. O PSP, assim como o CMM (*Capability Maturity Model*), é baseado no princípio da melhoria do processo. Enquanto o CMM é focado na melhoria da capacidade organizacional, o foco do PSP é o engenheiro.

1.3 Estrutura do trabalho

O Capítulo 1 introduz os assuntos abordados no presente trabalho.

O Capítulo 2 provê uma descrição da situação atual do desenvolvimento do *software*. Nesse capítulo, são apresentados dados de duas pesquisas, uma realizada nos Estados Unidos e a outra no Brasil.

Conceitos pertinentes ao trabalho e referências a algumas técnicas usadas são apresentados no Capítulo 3.

O Capítulo 4 fornece um resumo do PSP, na forma descrita no livro *A Discipline for Software Engineering*. Contém os dados e informações mínimas para executar de forma prática o método, mas de forma alguma substitui o texto original.

O Capítulo 5 trata do Estudo de Impacto do PSP, feito pelo SEI em 1997, bem como de outros estudos sobre o efeito do PSP em empresas.

O Capítulo 6 trata, de forma resumida, de diversas outras considerações importantes sobre o tema e tópicos atuais em discussão.

---

[3] Jones [Jones C. *apud* [CAS 99]] argumenta que muitas das falhas de software podem ser causadas por mal gerenciamento, em vez de falta de técnicas de desenvolvimento específicas. Entretanto, este ponto de vista não será discutido no presente trabalho.

O Capítulo 7 é a transcrição de *e-mails* recebidos de dois dos maiores especialistas de PSP no Brasil: Átila Belloquim e Jones Oliveira de Albuquerque.

No Capítulo 8 são feitos comentários e conclusões finais sobre o conteúdo do trabalho.

O Capítulo 9 comenta aqueles trabalhos sobre o PSP que o autor considerou mais relevantes e atuais, ou interessantes.

O apêndice A relaciona as tabelas mencionadas no trabalho, incluindo todas aquelas necessárias para o desenvolvimento do PSP. Optou-se por colocar todas as tabelas juntas para facilitar a consulta desse material.

2 Situação Atual: como se desenvolve *software* hoje

"Todo programa, quando acabado, estará obsoleto.
Todo programa custa mais e demora mais tempo.
Se um programa é útil, ele deve ser modificado.
Se um programa é inútil, ele deve ser documentado.
Os programas expandem de forma a encher toda a memória disponível.
O valor de um programa é proporcional ao peso de seus relatórios.
A complexidade de um programa cresce até exceder a capacidade do programador que deve mantê-lo.
Todo programa não-trivial possui pelo menos um *bug*.
Erros não-detectáveis são infinitos, ao contrário dos erros detectáveis que são, por definição, limitados.
Adicionar mais pessoas a um projeto de *software* atrasado o tornará ainda mais atrasado."
Leis de Murphy para a Programação, obtido de [JUN 00]

As organizações de *software* em todo o mundo empregam perto de 7 milhões de técnicos e geram anualmente uma receita de mais de 600 bilhões de dólares, com taxa de crescimento anual de mais de 25%. Metade desta receita é gerada por pacotes de *software* de uso geral, como os ERPs e a outra metade com o desenvolvimento de produtos específicos para clientes. A indústria de *software* é vista atualmente como um dos segmentos mais promissores, com um enorme potencial futuro. [COR 00]

Se considerarmos o desenvolvimento de *software* como um projeto, então, o foco desta indústria está na execução de projetos. Assumindo que, em média, cada projeto consome 7 pessoas/ano de esforço, então, a indústria de *software*, com seus 7 milhões de técnicos, executam algo em torno de 1 milhão de projetos por ano. Logo, desenvolver projetos de *software* eficientes é de fundamental importância para a indústria de *software* como um todo.

Os processos usados para desenvolver um projeto de *software* têm a maior importância na qualidade do *software* produzido e na produtividade alcançada pelo projeto. Por conseqüência, existe uma necessidade de melhorar os processos usados em uma organização para desenvolver projetos de *software*. [JAL 99, *apud* [COR 00]]

2.1 O Panorama atual da Indústria de *Software*

Informações colhidas de uma pesquisa americana realizada pelo instituto Standish Group em 1995 [STA 95], utilizando uma amostra de 365 companhias entrevistadas, representando 8.380 aplicações, visando identificar o âmbito das falhas de projetos de *software*, os fatores principais que causam falhas nos projetos de *software* e os ingredientes chave que podem reduzir falhas de projeto, forneceram os seguintes resultados: [COR 00a] e [STA 95]

- Os EUA gastam US$ 250 bilhões por ano em desenvolvimento de aplicação de tecnologia da informação em aproximadamente 175 mil projetos.
- 31% de todos os projetos são cancelados antes do seu término, representando um desperdício da ordem de US$ 81 bilhões.
- 53% de todos os projetos chegam ao final tendo custado 189% do valor estimado, representando US$ 59 bilhões em custo adicional, atrasam em até 222% da estimativa original além de serem entregues com apenas 61% das características originalmente especificadas.
- 16% são entregues no prazo e dentro do orçamento.

O custo destes fracassos e *bugs* (erros de *software*) são apenas a ponta do *iceberg*. Os custos de oportunidades perdidas não são mensuráveis, mas poderiam facilmente chegar a trilhões de dólares.

Perfil de Sucesso/Falha

O aspecto mais importante da pesquisa era descobrir por que os projetos falham. [STA 95] O Standish Group examinou gerentes executivos de TI para saber as suas opiniões sobre por que projetos têm sucesso (Tabela 2-1). As três razões principais que um projeto terá sucesso são: envolvimento de usuário, apoio da administração e uma declaração clara de exigências. Há outros critérios de sucesso, mas com estes três elementos, as chances de sucesso são muito maiores. Sem eles, a chance de fracassos aumenta dramaticamente.

Tabela 2-1 - Fatores de Sucesso de Projeto

Fatores de Sucesso de Projeto	% de Respostas
1. Envolvimento do usuário	15.9%
2. Apoio da Administração	13.9%
3. Declaração clara de exigências	13.0%
4. Planejamento próprio	9.6%
5. Expectativas realísticas	8.2%
6. Medidas de projetos menores	7.7%
7. Pessoal competente	7.2%
8. Propriedade	5.3%
9. Visão clara & objetivos	2.9%
10. Pessoal trabalhador, focalizado	2.4%
Outro	13.9%

Fonte: [STA 95]

Por outro lado, um dos mais importantes aspectos da pesquisa foi o levantamento dos fatores principais que tornam um projeto crítico (projetos que extrapolam prazo, custos e que são entregues com a funcionalidade prejudicada). De acordo com os entrevistados, as principais razões que levam a problemas de projeto são identificadas na Tabela 2-2, abaixo:

Tabela 2-2 - Fatores de Projetos Críticos

Fatores de Projetos Críticos	% de Respostas
1. Falta de Especificação do Usuário	12.8%
2. Requisitos Incompletos	12.3%
3. Mudança de Requisitos	11.8%
4. Falta de Apoio Executivo	7.5%
5. Tecnologia Imatura	7.0%
6. Falta de Recursos	6.4%
7. Expectativas Irreais	5.9%
8. Objetivos obscuros	5.3%
9. Tempo irreal	4.3%
10. Tecnologia nova	3.7%
11. Outros	23.0%

Fonte: [STA 95]

Outro achado fundamental da pesquisa é que uma porcentagem alta de gerentes executivos acreditava que havia mais fracassos de projeto na ocasião do que há cinco e dez anos atrás. Isto apesar do fato de que a tecnologia teve tempo para amadurecer.

Tabela 2-3 - Fracassos de Projeto

	Que 5 Anos Atrás	Que 10 Anos Atrás
Significativamente Mais Fracassos	27%	17%
Um pouco Mais Fracassos	21%	29%
Nenhuma Mudança	11%	23%
Um pouco Menos Fracassos	19%	23%
Significativamente Menos Fracassos	22%	8%

Fonte: [STA 95]

O caminho para o sucesso

Segundo o Standish Group, a situação atual dos projetos de *software* é verdadeiramente preocupante.[STA 00] Para trazer um pouco de ordem ao caos, é preciso investigar, estudar e compartilhar as causas de cada fracasso de *software*.

Mais diretamente ligado ao objeto de estudo do presente trabalho, a pesquisa do Standish Group indica que prazos menores, com entrega de componentes de *software*, prematura e freqüentemente, aumentará a taxa de sucesso. Segundo o Grupo, prazos mais curtos resultam em um processo interativo de projeto, protótipo, desenvolvimento, testes e desdobre de pequenos elementos. Este processo é conhecido como "*software* crescente", ao invés do velho conceito de "*software* em desenvolvimento". O *Software* Crescente cativa o usuário mais cedo, cada componente tem um proprietário ou um pequeno conjunto de proprietários, e as expectativas são realisticamente fixas. Além disso, cada componente de *software* tem uma declaração e um conjunto de objetivos claros e precisos. **Componentes de *software* e projetos pequenos** tendem a ser menos complexos. Fazer os projetos mais simples é um empenho que vale a pena porque a complexidade só causa confusão e aumento de custo. [STA 00]

Possivelmente, a inabilidade para trabalhar mais efetivamente com usuários e entender melhor as suas necessidades (requisitos) tem sido uma das causas principais das falhas de *software*. Esta deficiência, aliada à desordem de muitos ambientes de desenvolvimento, que muitas vezes buscam a melhoria da qualidade de seus produtos, mas não investem em processos básicos, como por exemplo, a documentação dos seus sistemas, é que produz esta situação de caos.

"Parece-me que o ingrediente principal para o nosso trem engrenar seja a disciplina. Precisamos de um processo consistente para seguir e disciplina para realizá-lo", diz [COR 00a].

2.2 Qualidade e Produtividade no Setor de *Software* Brasileiro

O Governo brasileiro vem realizando pesquisas diretas junto a empresas desenvolvedoras de *software* no Brasil, objetivando acompanhar a evolução desse setor quanto a aspectos do planejamento estratégico nas empresas, sistemas da qualidade e certificação, qualidade dos processos e dos produtos, gestão da força de trabalho, relacionamento das empresas com seus clientes, métodos, ferramentas e procedimentos para a qualidade dos produtos de *software* no país. [MCT 00]

Em 1999, em sua quarta edição, a pesquisa encerrou-se com a participação de 446 empresas. A estrutura e o conteúdo da pesquisa são mostrados na Tabela 2-4.

Tabela 2-4 - Estrutura da Pesquisa "Qualidade e Produtividade no Setor de *Software* Brasileiro"

Categorias	**Principais Itens**
Caracterização das Empresas	Atividades em Tecnologia da Informação
	Atividades no desenvolvimento de *Software*
	Localização geográfica e Idade da empresa
	Porte por força de trabalho e por comercialização
Qualificação Profissional	Formação acadêmica da força de trabalho
	Profissionais certificados em qualidade
	Promoção da atualização profissional
	Treinamento
Terceirização de Serviços	Análise e Programação
	Marketing e Vendas
Caracterização do *Software*	Produtos desenvolvidos
Sistemas da Qualidade	Planejamento estratégico
	Metas ou diretrizes para a qualidade
	Indicadores e custos da qualidade
	Programas da Qualidade Total
Qualidade de Processos	Certificação ISO 9000
	CMM, SPICE, ISO/IEC 12207 - conhecimento e uso
Qualidade de Produtos	ISO/IEC 9126, ISO/IEC12119 - conhecimento e uso
	Avaliação de produtos baseada em normas
Gestão da Força de Trabalho	Métodos para apoiar a participação
	Avaliação de desempenho

Categorias	Principais Itens
	Pesquisas de satisfação
Relacionamento com os Clientes	Pesquisas de expectativa e de satisfação
	Estruturas de atendimento
	Uso de dados na revisão ou especificação de novos produtos e serviços
Procedimentos para Qualidade	Engenharia de *Software* Métodos para prevenção de defeitos Métodos para detecção de defeitos
	Ferramentas de desenvolvimento
	Documentação
Produtividade	Métodos para medição de processos

Fonte: MCT/SEPIN [MCT 00]

A coordenação, execução e divulgação das pesquisas da Qualidade e Produtividade são responsabilidade da SEPIN através da Divisão de Sistemas de Informação sobre Informática, no âmbito do Subcomitê Setorial da Qualidade e Produtividade em *Software*, do Programa Brasileiro da Qualidade e Produtividade em *Software* – SSQP/SW-PBQP.

Encontram-se definidos 28 "Indicadores e Metas da Qualidade e Produtividade em *Software*", que fazem parte integrante do Termo de Referência do SSQP/SW-PBQP, sendo 2 da categoria de Conscientização e Motivação, 8 de Métodos de Gestão, 4 de Recursos Humanos, 3 de Serviços Tecnológicos, 4 de Articulação Institucional, 3 de Tecnologia de *Software* (onde são acompanhados 25 métodos e ferramentas de Engenharia de *Software*) e 4 de *Marketing* de *Software*.

Resultados para alguns indicadores selecionados:

Tabela 2-5 - Indicadores em *Software*

Indicadores Selecionados	**1995**	**1997**	**1999**
Percentual de empresas com programas da qualidade total, sistemas da qualidade ou similar implantados	11%	18%	26%
Percentual de empresas com sistema da qualidade certificado (ISO 9001 e ISO 9002)	2%	8%	17%
Número de empresas com *software* explicitado no escopo do certificado de qualidade (ISO)	-	16	35
Percentual de empresas que conhecem e usam o modelo CMM (*Capability Maturity Model*)	3%	5%	10%
Percentual de empresas que usam a Norma ISO/IEC 9126 para avaliação de produtos	-	7%	10%

Indicadores Selecionados	**1995**	**1997**	**1999**
Número de profissionais certificados em qualidade em empresas que atuam no segmento de *software* (certificação ASQ, *Lead Assessor*, pós-graduação *lato sensu* e *stricto sensu* em gestão da qualidade)	390	366	823
Percentual dos investimentos anuais em treinamento para melhoria da qualidade sobre a comercialização bruta proveniente de *software*	3%	2,5%	2,3%
Percentual de empresas que utilizam, de forma sistemática, dados de pesquisa ou de reclamações na revisão de projetos ou na especificação de novos produtos	41%	44%	44%
Percentual de empresas que atuam no segmento de *software* e realizam, de forma sistemática, pesquisas de satisfação dos clientes	19%	25%	29%

Fonte: MCT/SEPIN [MCT 00]

Segundo a SEPIN, os resultados obtidos nas pesquisas revelam um aumento crescente da conscientização pela qualidade nas empresas de *software* no Brasil, com exigências cada vez maiores de clientes e usuários. Entretanto, apesar da crescente preocupação com a qualidade, é possível notar que a porcentagem de empresas que investem efetivamente em qualidade é relativamente baixa.

Confrontando a pesquisa da SEPIN com aquela do Standish Group, é possível verificar uma grande diferença com relação à filosofia empregada: enquanto que o governo brasileiro optou por analisar a qualidade consultando as empresas *produtoras* de *software*, a empresa americana realizou a sua pesquisa junto aos *usuários* finais dos produtos, empresas que tinham departamentos de Sistemas de Informação instituídos. Considera-se mais realista e efetiva a abordagem utilizada pelo Standish Group, já que as definições da qualidade devem considerar às perspectivas dos usuários. [HUM 95] A qualidade do *software* pode ser definida como o *software* que satisfaz as necessidades dos usuários.

3 Conceitos

> "Não é o bastante fazer o melhor: você tem que saber o que fazer, e ENTÃO fazer o melhor. "
> W. Edwards Deming[4]

O desenvolvimento de um artefato pode ser conduzido de forma artesanal, por um processo de tentativa-e-erro, manipulando-se diretamente o material com o qual o produto será construído. Os erros são identificados através da avaliação experimental da qualidade do produto. A avaliação deve verificar se o produto está funcionando adequadamente, se ele é útil aos seus usuários e várias outras qualidades.

Este processo artesanal pode evoluir através da utilização de ferramentas específicas e de técnicas desenvolvidas a partir de experiências anteriores. Este processo pode ainda evoluir e incorporar outras atividades. O *design* consiste na atividade de conceber e descrever o produto a ser construído. O design permite uma visualização antecipada do produto final permitindo que se possa fazer alguma avaliação antes da sua construção. Para que o produto tenha sucesso ele deve estar adequado às necessidades do usuário. Atividades de análise de requisitos devem anteceder o design e a construção do artefato para que estes objetivos sejam atingidos.

Com tantas atividades é preciso um **modelo do processo** que descreva em qual momento cada uma será realizada. Análise, especificação, *design*, implementação e avaliação devem ser realizados no momento adequado. Para cada etapa do processo, métodos devem descrever com detalhes como estas atividades devem ser realizadas.

3.1 Engenharia de *Software*

A disciplina que vai ajudar a entender o processo de desenvolvimento de *software* é a Engenharia de *Software*. É através dela que se pode chegar à qualidade. Existe, entretanto, um grande problema a ser resolvido: tecnicamente, **ela não existe**.[JUN 00]

O problema é que, para que uma disciplina seja considerada realmente uma Engenharia, é necessário atender a alguns requisitos básicos que a Engenharia de *Software*, pelos menos até agora, não atende. A Engenharia de *Software* falha principalmente no que diz respeito à adequação do custo-benefício e à aplicação, em toda a sua extensão, de conhecimentos científicos. Atualmente, estes requisitos são atendidos apenas em parte.[JUN 00]

---

[4] Extraído de *Software Engineering Proverbs* - Tom Van Vleck , http://www.multicians.org/thvv/proverbs.html (05/11/2000).

É necessário definir, portanto, o que é exatamente a Engenharia de *Software*. Algumas tentativas de definição: [JUN 00]

> *"...é a disciplina que integra métodos, ferramentas e procedimentos para o desenvolvimento de software para computadores."*
>
> *"...é uma coleção de processos de gerenciamento, ferramental de software e atividades de projeto para o desenvolvimento de software. "*
>
> *"...é um termo usado para referir-se a modelos de ciclo de vida, metodologias de rotina, técnicas de estimativa de custo, estruturas de documentação, ferramentas de gerenciamento de configuração, técnicas de garantia de qualidade e outras técnicas de padronização da atividade de produção de software."*

3.2 Qualidade do Produto x Qualidade do Processo

A maioria dos métodos para melhorar a qualidade (análise e metodologias de projeto, linguagens, revisões, inspeções, testes) concentra-se no produto. Os erros são corrigidos, mas pouco esforço é investido em identificar por que eles foram introduzidos e em assegurar que problemas semelhantes não ocorram periodicamente. [CAS 99]

Uma das evoluções mais importantes no estudo da qualidade está em notar que a **qualidade do produto** é algo bom, mas que **qualidade do processo** de produção é ainda mais importante. [JUN 00]

Esta descoberta aconteceu durante a própria evolução dos conceitos de qualidade, ao longo dos anos. A tabela abaixo mostra como aconteceu esta evolução:

Tabela 3-1 - Evolução dos conceitos de qualidade

Inspeção pós-produção	Avalia o produto final, depois de pronto	1900
Controle estatístico da produção	Avalia os subprodutos das etapas de produção	1940
Procedimento de produção	Avalia todo o procedimento de produção	1950
Educação das pessoas	Avalia as pessoas envolvidas no processo	1960
Otimização dos processos	Avalia e otimiza cada processo	1970
Projeto robusto	Avalia o projeto de produção	1980
Engenharia simultânea	Avalia a própria concepção do produto	1990

Fonte: [JUN 00]

Hoje em dia, é possível consultar normas e padrões tanto para produtos quanto para processos. Os certificados mais valiosos são aqueles que garantem a qualidade do processo de produção de um produto e não aqueles que simplesmente certificam o produto. Entretanto, é comum encontrar empresas que perseguem os dois tipos de padrão de qualidade.

3.3 Qualidades do *software*

O *software* como um produto deve ter qualidade. Algumas das qualidades que podem ser avaliadas são: a corretude, validade, robustez, confiabilidade, eficiência, usabilidade, manutenibilidade, evolutibilidade, portabilidade, interoperabilidade e reusabilidade.

Entretanto, é preciso avaliar tanto a qualidade do produto em si com a do processo de desenvolvimento. Assim, os fatores de qualidade de *software* podem ser classificados como do ***processo*** ou do ***produto***. Os fatores de qualidade do processo visam assegurar a qualidade do desenvolvimento do produto em todas as suas etapas. Os fatores relacionados ao produto permitem analisar a qualidade do *software* em si.[LEI 00]

Cada um destes fatores precisam ser considerados para garantir a qualidade do *software*. Para isto, deve-se utilizar técnicas para avaliação e análise de cada um desses fatores.

3.4 Qualidade do Processo de *Software*

Os estudos sobre qualidade mais recentes são na sua maioria voltados para o melhoramento do processo de desenvolvimento de *software*. Não porque a qualidade do produto não seja importante. Mas o fato é que, ao garantir a qualidade do processo, já se está dando um grande passo para garantir também a qualidade do produto.

O estudo da Qualidade do Processo de *Software* é uma área ligada diretamente à Engenharia de *Software*. O estudo de um ajuda a entender e aprimorar o outro. Em ambas as disciplinas, estudam-se modelos do processo de desenvolvimento de *software*. Estes modelos são uma tentativa de explicar em detalhes **como** se desenvolve um *software*, quais são as etapas envolvidas visando melhor compreender cada pequena tarefa envolvida no desenvolvimento.

3.5 O Processo

Elaborar um **processo de desenvolvimento de *software*** significa determinar de forma precisa e detalhada quem faz o que, quando e como. Um processo pode ser visto como uma instância de um método com suas técnicas e ferramentas associadas, elaborado durante a etapa de planejamento, no qual as atividades que o compõem foram alocadas aos membros da equipe de desenvolvimento, com prazos definidos e métricas para se avaliar como elas estão sendo realizadas.[LEI 00]

Enquanto um método é algo teórico, o processo deve determinar ações práticas a serem realizadas pela equipe, como prazos definidos. O processo é o resultado do planejamento e precisa ser gerenciado no decorrer de sua execução.

Processo

No ciclo de vida do *software* identificamos três fases:

- **Definição**
- **Desenvolvimento**
- **Operação**

Na fase de **definição** os requisitos do *software* são determinados, a sua viabilidade é estudada e o planejamento das atividades é elaborado.

Na fase de **desenvolvimento** são realizadas as atividades destinadas a produção do *software*. Ela envolve atividades de concepção, especificação, design da interface, prototipação (do inglês *prototyping*, traduzido também por prototipagem), design da arquitetura, codificação e verificação, dentre outras.

Na fase de **operação** o sistema deverá efetivamente ser utilizado pelos seus usuários produzindo os resultados desejados. Nesta fase devem ocorrer as atividades de manutenção, seja para que se façam correções, ou seja para a sua evolução, isto é, para que o *software* satisfaça novos requisitos.[LEI 00]

Como normalmente o *software* está inserido num contexto mais amplo - o sistema - a fase de definição do *software* está inserida na definição do sistema. Definir o sistema é definir todos os seus componentes

Também na fase de definição, caso o *software* seja viável, deve ser feito o planejamento de como o desenvolvimento será conduzido, isto é, deve-se elaborar um **processo de desenvolvimento**. O planejamento não necessariamente precisa ser feito completamente nesta fase. O resultado do planejamento é uma descrição precisa do **processo de desenvolvimento de *software***. [LEI 00]

Modelos do processo de desenvolvimento

Um modelo para um processo de desenvolvimento é uma proposta teórica que junto com o planejamento deve determinar quais atividades devem ser realizadas, quando, como e por quem. Alguns dos mais conhecidos modelos do processo de desenvolvimento são o modelo Cascata, o modelo Evolutivo ou Incremental, o modelo Espiral e o modelo Transformação.

3.6 Execução de projetos baseados na abordagem de processos

Um projeto de desenvolvimento de *software* visa a construção de um produto de *software* que satisfaça as necessidades do cliente e seja desenvolvido e entregue dentro do custo e prazo especificado. Em outras palavras, as três principais características de um projeto são: custo, planejamento e qualidade, onde qualidade representa quão bom o produto é e satisfaz o cliente. [COR 00]

Um projeto é um sucesso quando atinge ou excede as expectativas destas três características – custo, planejamento e qualidade. Porém, a indústria de *software* pode citar muitos exemplos de projetos que fracassaram. A situação vem melhorando com o passar dos anos mas ainda existem muitos projetos fracassando em termos de cronograma, custo e qualidade. Uma pesquisa utilizando dados de projetos [PUT 97 *apud* [COR 00]] mostra que 1/3 destes projetos tem custo e cronograma além de 125% da estimativa inicial. Exemplos de projetos que saíram do controle têm sido documentados. [GLA 98, *apud* [COR 00]] e [STA 95]

As possíveis razões para falhas de projeto incluem estimativas impróprias, falta de gerenciamento de requisitos, fraco gerenciamento de projeto, gerenciamento de risco impróprio, pobres soluções de engenharia. Muitas destas razões podem ser combinadas em uma categoria chamada "falha do processo". Um projeto de *software* falha, muitas vezes, porque o processo seguido não é apropriado. As maiores razões para o descontrole são: objetivos pouco claros, planejamento ruim, utilização de novas tecnologias, falta de uma metodologia de gerenciamento do projeto e pessoal insuficiente [GLA 98]. Todas estas razões podem ser consideradas falha do processo. Para um projeto ter sucesso, um parâmetro chave é o conjunto de processos utilizados. Se processos apropriados são utilizados, as chances do projeto ter sucesso são extremamente altas.

Alta produtividade pode gerar redução de custo e minimizar o tempo do projeto. Qualidade e produtividade são as metas de toda a organização, porém, ainda existe um objetivo mais primordial das organizações. É a previsibilidade. Não é suficiente que um projeto tenha alta qualidade e produtividade. A organização também quer prever qual será a qualidade e produtividade atingida. Se não se consegue ter previsibilidade, é

porque não se consegue realizar estimativas apropriadas e isto é o mesmo que caminhar no escuro. [COR 00]

Como o processo tem o maior efeito sobre qualidade e produtividade, um modo de melhorar este par está na melhoria dos processos usados pela organização. [COR 00]

3.7 A abordagem do PSP

Um método é um modelo genérico de processo de desenvolvimento. Ele é uma proposta teórica para que um processo possa ser elaborado e colocado em prática.

O capítulo seguinte apresenta o PSP, um método que busca a melhoria da qualidade do trabalho do Engenheiro de *Software*, através da definição precisa e qualitativa de um processo pessoal.

O PSP baseia-se nos conceitos clássicos de qualidade. E entre os conceitos mais importantes de qualidade encontra-se o de *Garantia de Qualidade*, como oposto ao *Controle de Qualidade*. Enquanto o *controle* de qualidade procura *encontrar defeitos* no produto acabado, a *garantia* de qualidade procura garantir que, em cada etapa da fabricação do produto, defeitos não sejam injetados.[BEL 00]

4 O PSP

> "Não existe nenhum estudante na classe que tenha esquecido que 250 engenheiros gastaram um ano inteiro para remover 30.000 defeitos do Windows NT, uma média de 16 horas por defeito!" [WIL 97]

Este capítulo[5] descreve o PSP, sua lógica, sua estrutura e seus conceitos. Aprofunda-se nos conceitos de processo e descreve o papel do processo no desenvolvimento de *software*. Relaciona os problemas de desenvolvimento de *software* industrial aos assuntos de disciplina profissional e descreve como usar métodos disciplinados para melhorar o desempenho pessoal do engenheiro de *software*.

4.1 Introdução

4.1.1 O que é o PSP?

Watts Humphrey define o PSP como sendo um processo de auto-melhoria projetado para ajudar o engenheiro de *software* a controlar, administrar e melhorar o modo como ele trabalha. O PSP é uma estrutura de formulários, diretrizes e procedimentos para o desenvolvimento de *software*. Corretamente usado, o PSP provê os dados históricos que o engenheiro de *software* precisa para fazer e conhecer melhor seus compromissos e torna os elementos rotineiros de seu trabalho mais previsíveis e mais eficientes.

O propósito exclusivo do PSP é ajudar o engenheiro de *software* a fazer um trabalho melhor. É uma ferramenta poderosa para ser usada de muitas formas, como administrar o trabalho, avaliar o próprio talento e construir habilidades. Pode ajudar a planejar melhor, descobrir precisamente o próprio desempenho e medir a qualidade dos produtos.

Entretanto, Humphrey salienta que o PSP não é uma resposta mágica a todos os problemas do engenheiro de *software*. Embora possa sugerir onde e como melhorar, o engenheiro tem que fazer as melhorias por si próprio.

---

[5] Este capítulo é um resumo de [HUM 95]. Procurou-se apresentar o PSP de forma direta e prática, com o conteúdo mínimo necessário para o entendimento da metodologia proposta.

O PSP é introduzido com um curso que consiste de quinze leituras, dez tarefas de programação e cinco relatórios[6]. O curso é ensinado efetivamente ao nível de diplomados. Ele também é ensinado na indústria de *software*.

Esta seção documentará a estrutura incremental do Processo de *Software* Pessoal. Também incluirá uma descrição dos formulários do PSP e a sua aplicabilidade no processo de desenvolvimento de *software*.

4.1.2 Motivações do PSP

Os métodos intuitivos de desenvolvimento de *software* geralmente usados hoje, só são aceitáveis porque não existem alternativas. A prática atual está mais próxima de uma arte do que de uma disciplina de engenharia. Os profissionais geralmente desenvolvem seus próprios métodos e técnicas privadas. A maioria dos produtos de *software* já concluídos podem funcionar, mas normalmente só depois de extensos testes e reparos. De um ponto de vista científico, o processo é imprevisível.

Esta situação fica crítica quando a contribuição de cada indivíduo é exclusivamente importante. Uma orquestra sinfônica ilustra melhor esta idéia. Enquanto que o desempenho global da orquestra é uma mistura cuidadosa de muitos instrumentos, cada músico é um contribuinte altamente competente e disciplinado. Artistas individuais se salientam ocasionalmente, mas a orquestra inteira é muito mais que a soma destas partes, e uma única nota estragada de qualquer indivíduo poderia danificar o desempenho inteiro.

Na maioria das profissões, competência requer proficiência demonstrada com métodos estabelecidos. Não é uma questão de criatividade *versus* habilidade, porque freqüentemente trabalho criativo simplesmente não é possível até que a pessoa domine as técnicas básicas. Disciplinas bem fundadas encapsulam anos de conhecimento e experiência. Profissionais iniciantes, como na física de alta-energia e na cirurgia do cérebro, têm que demonstrar proficiência em muitas técnicas antes de lhes permitirem executar até mesmo os procedimentos mais rotineiros. A habilidade sem defeito, uma vez adquirida, aumenta a criatividade. Um profissional qualificado em um campo pode superar até mesmo o leigo mais brilhante, mas destreinado.

Uma organização disciplinada de engenharia de *software* terá práticas bem definidas. Seus profissionais usarão essas práticas, monitorar-se-ão e se esforçarão para melhorar seu desempenho e se sentirão responsáveis pelo controle de qualidade. Terão a confiança e os dados requeridos para resistir a demandas desarrazoadas de compromissos.

---

[6] O curso e todo o material está contido em [HUM 95] .

4.1.3 Processo de *Software*

O *processo de software* é a sucessão de passos necessários para o desenvolvimento e manutenção de *software*. Uma *definição* de processo de *software* é uma descrição deste processo. Quando corretamente projetada e apresentada, a definição guia os engenheiros de *software* sobre como eles trabalham.

Mais especificamente, o processo de *software* inicia a estrutura técnica e administrativa para a aplicação de métodos, ferramentas e pessoas para a tarefa de programar, enquanto que a definição de processo identifica papéis e especifica tarefas.

Processos definidos provêem os seguintes benefícios:

- habilitam a efetiva comunicação sobre o processo entre usuários, desenvolvedores, gerentes, clientes e pesquisadores.
- aumentam o entendimento da administração, provêem uma base precisa para a automatização do processo e facilitam a mobilidade de pessoal.
- facilitam o reuso de processos. O desenvolvimento de processos é caro e consumidor de tempo. Poucos grupos de projeto podem dispor de tempo ou recursos para definir completamente o modo como eles trabalharão. Eles podem economizar ambos, usando os elementos reutilizáveis padrões que um processo definido provê.
- suportam a avaliação de processo provendo meios efetivos para aprendizado de processos e uma fundação sólida para a melhoria de processo.
- ajudam a administração dos processos. A administração efetiva requer planos claros e um modo preciso e quantificado para medir seu estado. Processos definidos provêem, assim, uma estrutura.

4.1.4 Maturidade de processo

Os processos para o desenvolvimento de *software* em larga escala podem ser grandes e complexos. Assim, eles são freqüentemente difíceis de definir, de compreender e de introduzir nas organizações. Esta é a razão pela qual a estrutura de maturidade de processo de *software* foi desenvolvida[7]. Esta estrutura é um modo para as organizações determinarem as capacidades dos seus processos atuais de estabelecerem prioridades para melhorias. Faz-se isto constituindo e definindo cinco níveis progressivos de capacidades de processo mais maduras:

[7] Humphrey é o criador do CMM.

1. Inicial: o processo de *software* é caracterizado como *ad hoc* e até mesmo caótico ocasionalmente. Poucos processos estão definidos e o sucesso depende do esforço individual.

2. Repetível: são estabelecidos processos básicos de administração de projeto para identificar custos, horários e funcionalidade. A disciplina de processo necessária está em repetir sucessos em projetos com aplicações semelhantes.

3. Definido: o processo de *software*, para a administração e para as atividades de engenharia, é documentado, unificado e integrado em um processo de *software* padrão. Todos os projetos usam uma versão aprovada e costurada do processo de *software* padrão da organização para desenvolver e manter *software*.

4. Administrado: detalhadas medidas de processo de *software* e qualidade de produto são coletadas. O processo de *software* e os produtos são quantitativamente compreendidos e controlados.

5. Aperfeiçoado: a melhoria contínua de processo é habilitada através de avaliação quantitativa do processo e da condução de idéias e tecnologias inovadoras.

O Instituto de Engenharia de *Software* (SEI), trabalhando com as principais organizações de *software* norte-americanas, refinou estes níveis de definições e as suas práticas no Modelo de Maturidade de Capacidade (CMM) para *Software*.[PAU 93] A cada nível, *key process áreas* (KPAs) provêem metas e exemplos práticos.

O PSP tem uma estrutura de maturidade como o CMM. A Figura 4-1 mostra o CMM com as KPAs que são resolvidas pelo menos parcialmente pelo PSP, mostradas em itálico e com um asterisco. Alguns itens do CMM são excluídos por não se adequarem à aplicação no nível individual.

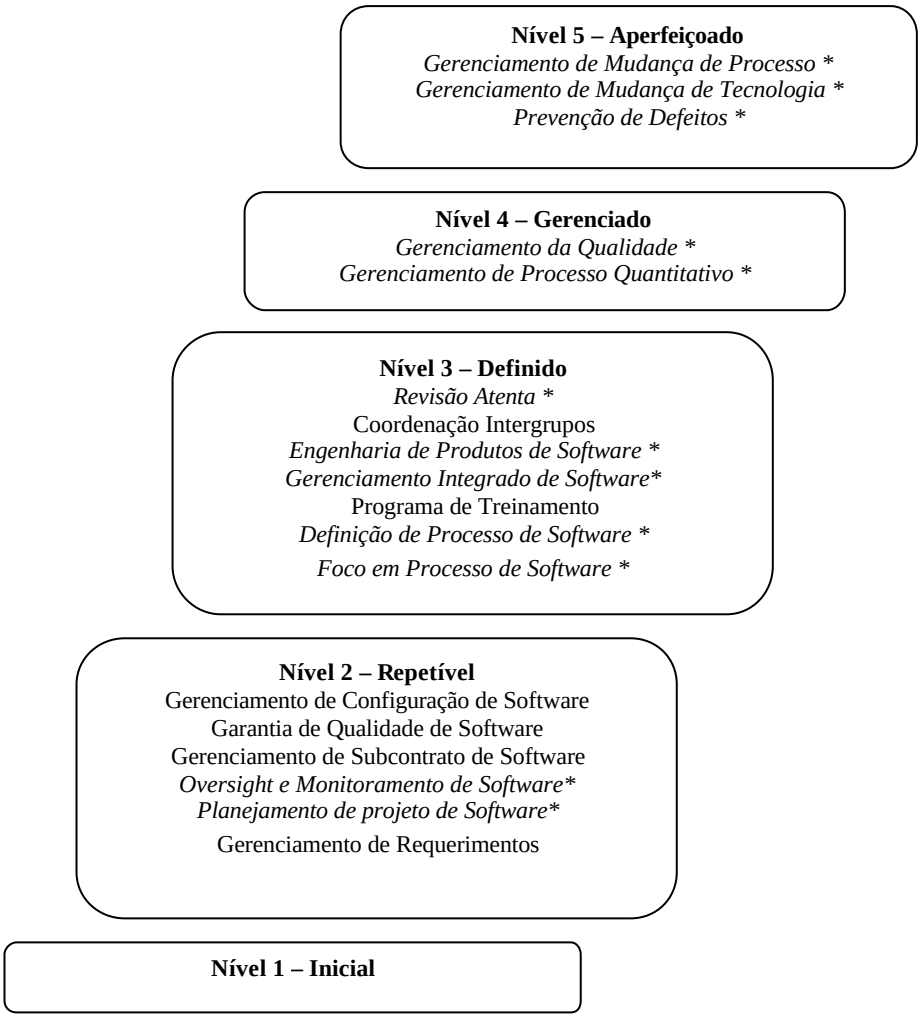

Figura 4-1 - O CMM e o PSP

O CMM por si só, possibilita ao engenheiro de *software* que faça um bom trabalho, mas não garante este, pois o trabalho bem feito depende em muito do uso efetivo de práticas pessoais do engenheiro. É necessário, para tal, que os engenheiros saibam como produzir produtos de qualidade usando princípios individuais de aperfeiçoamento de processo. É aí que entra o PSP. [CRI 00]

Entretanto, para serem efetivos, os engenheiros precisam do apoio de um ambiente eficiente e disciplinado, o que significa que o PSP será mais efetivo se usado em organizações próximas ou acima do nível 2 do CMM.

Com isso em vista, conclui-se que o CMM e o PSP são mutuamente dependentes. O CMM provê um ambiente de apoio que os engenheiros precisam para fazer seu trabalho e o PSP os equipa para fazer trabalhos de alta qualidade e participar do aperfeiçoamento do processo organizacional. [CRI 00]

4.1.5 A lógica do PSP

O PSP foi derivado de princípios provados em outros campos. A lógica para o PSP é a seguinte:

- Os profissionais de *software* entenderão melhor o que eles fazem se eles definirem, medirem e monitorarem o seu trabalho.
- Eles terão então uma estrutura de processo definida e critérios mensuráveis para avaliar e aprender através de suas próprias experiências e das dos outros.
- Com este conhecimento e experiência, eles podem selecionar os métodos e práticas que melhor se adaptem as suas tarefas e habilidades particulares.
- Usando um conjunto padronizado ordenadamente, constantemente praticado, e práticas pessoais de alta qualidade, eles serão os membros mais efetivos dos seus grupos de desenvolvimento e projeto.

Esta lógica é baseada em cinco princípios. Estes princípios são resumidos abaixo:

1. Um processo definido e estruturado pode melhorar a eficiência do trabalho.
2. Processos pessoais definidos devem se ajustar convenientemente às habilidades individuais e preferências de cada engenheiro de *software*.
3. Para os profissionais estarem confortáveis com um processo definido, eles devem ser envolvidos em sua definição.
4. Como a perícia e as habilidades dos profissionais evoluem, os seus processos deveriam fazer o mesmo.
5. A melhoria contínua de processo é alcançada através de rápido e explícito *feedback*.

4.1.6 Produtividade e PSP

Com relação à produtividade, Humphrey argumenta que, no início, ao escrever os primeiros programas, a produtividade do estudante do PSP deve recuar. À medida que se ganha experiência com o PSP, porém, essa produtividade deveria ser maior do que a inicial. Entretanto, de acordo com o estudo de impacto do PSP, discutido mais adiante, verificou-se que o PSP não altera os índices de produtividade.

4.1.7 Caveats

O trabalho desenvolvido por Humphrey concentra-se no projeto, codificação e fases de testes de desenvolvimento de *software*. Porém, o PSP não se aplica só a eles, mas também a quase qualquer outro aspecto do processo de *software*, inclusive especificação de requerimentos, manutenção de produto, planejamento de testes e desenvolvimento de documentação.

Projeto detalhado, código e fases de testes de unidades do processo de *software* são partes críticas do processo de desenvolvimento de *software*. De acordo com Boehm, consomem de 59 a 68 por cento do custo de desenvolvimento na maioria organizações de *software*. [BOH 81, *apud* [HUM 95]] Além disso, elas também são a parte do processo que produz o produto corrente. Isto não significa que nada mais é importante, apenas que estas fases devem ser bem feitas ou o resto não importará.

Cronograma, custo e problemas de qualidade de desenvolvimento de código tipicamente custam muito às organizações. Enquanto que os custos de testes são óbvios, custos de instalação e documentação não estão claramente relacionados à qualidade do código. Para muitos produtos, muita documentação é requerida, porque o produto é difícil de entender ou instalar ou para consertos ou atualizações posteriores. Muito desta documentação não seria necessária se o código fosse inicialmente de alta qualidade. Assim a organização de desenvolvimento e a organização do usuário seriam auxiliadas por uma melhor qualidade do código.

4.1.8 Os níveis do PSP

A implantação do PSP é dividida em 7 níveis, conforme pode ser visto na Figura 4-2. Esta implantação é feita de maneira incremental. Os níveis superiores adicionam características aos níveis já implantados. Isto minimiza o impacto da mudança no processo do engenheiro, no qual ele somente tem que adaptar novas técnicas as já existentes. Os níveis do PSP são descritos de forma sumária abaixo. Mais adiante, o assunto será retomado e aprofundado.

- PSP0 - a base: o primeiro passo no PSP é estabelecer uma base (*baseline*), que inclui algumas medidas básicas e um relatório. Esta *baseline* provê uma base consistente para medir o progresso e uma fundação definida sobre a qual melhorar. O PSP0 deve ser o processo normal que se usa para escrever *software*s, mas acrescido com medições.
- PSP1- o processo de planejamento pessoal: o PSP1 adiciona o planejamento de passos ao PSP0. O incremento inicial adiciona um relatório de testes e estimativa de recursos e tamanho.
- PSP1.1: no PSP1.1, são introduzidos planejamento de horário e tarefas.

- PSP2 - o processo de administração da qualidade pessoal: para administrar os seus defeitos, o desenvolvedor tem que saber quantos faz. O PSP2 acrescenta técnicas de revisão ao PSP1, para ajudar a achá-los no início, quando forem menos caros para resolver. Faz-se isto juntando e analisando os defeitos achados na compilação e nos testes dos primeiros programas. Com estes dados, o desenvolvedor pode estabelecer listas de conferência de revisão e fazer suas próprias avaliações de qualidade de processo.
- PSP2.1: o PSP2.1 estabelece critérios de perfeição de projeto e examina várias técnicas de verificação e consistência de projeto.
- PSP3: a estratégia do PSP3 é subdividir um programa maior em pedaços do tamanho requerido pelo PSP2. A primeira construção é um módulo básico ou núcleo, que aumenta em ciclos de iteração. Em cada repetição, faz-se um PSP2 completo, incluindo projeto, compilação e testes. Assim, o PSP3 é satisfatório para programas de até várias mil LOC (KLOC). O processo PSP3 cíclico efetivamente escala programas grandes contanto que cada incremento sucessivo seja de alta qualidade.

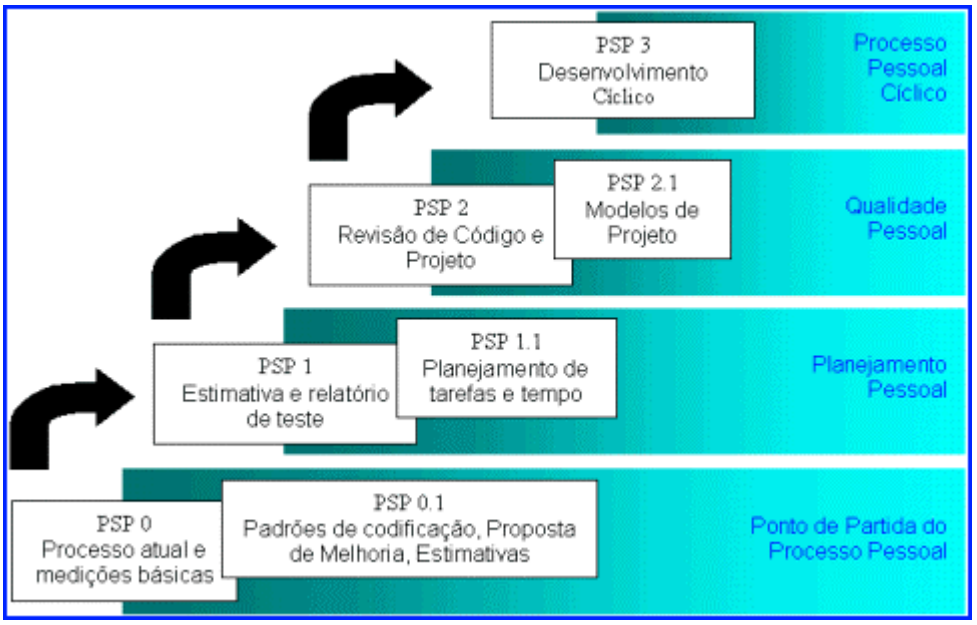

Figura 4-2 - Níveis do PSP

A seguir, os níveis do PSP são examinados mais detalhadamente. A apresentação dos níveis é feita da seguinte forma:

- primeiro são abordados e discutidos conceitos referentes ao nível em questão;
- a seguir, nas seções intituladas “Conteúdos do Processo”, o próprio processo é descrito;
- por fim, são propostos os exercícios referentes à fase.

4.2 PSP0 - O Processo Básico

Esta seção define o processo inicial usado pelos primeiros exercícios. Ao se definir um processo pessoal, começa-se a pensar em seus termos. Tarefas abstratas são estruturadas e sujeitas a análise racional. Adquire-se uma estrutura para medidas e um foco para melhorias

O PSP0 é a base para as ampliações de processo introduzidas nos níveis seguintes. Ele é mostrado na Fig. 4-3. Os *scripts* guiam através dos passos do processo, os *logs* ajudam a registrar os dados dos processos e o *resumo de plano* provê um modo conveniente para registrar e informar os resultados[8]. Este processo provê:

- uma estrutura conveniente para fazer tarefas em pequena escala: o que fazer primeiro? O que fazer depois? E assim por diante.
- uma estrutura para medir estas tarefas: um processo definido permite juntar dados do tempo gasto em cada tarefa de *software* e monitorar o número de defeitos introduzidos e removidos em cada passo de processo. Estes dados ajudam a analisar o processo, entender os erros e melhorá-lo.
- uma base para a melhoria de processo: se não se sabe exatamente o que se está fazendo, é difícil melhorar este trabalho.

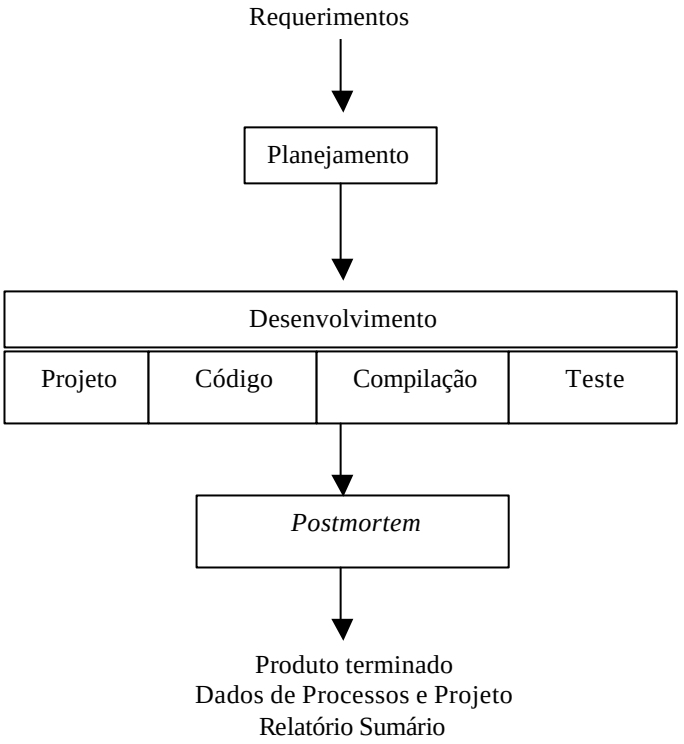

Figura 4-3 - O PSP0

[8] As tabelas, formulários e logs são mostrados no apêndice A, com exceção daqueles necessários ao entendimento do texto.

4.2.1 Os elementos do processo

Primeiro, no passo de planejamento, é produzido um plano para fazer o trabalho. Depois, é feito o desenvolvimento do *software*. No fim, no passo de *Postmortem*, é feita uma comparação do desempenho atual com o planejado, são registrados os dados de processo e é produzido um relatório sumário. Embora estes passos de planejamento e *Postmortem* possam parecer desnecessários ao se escrever programas pequenos, eles são essenciais para se construir um processo pessoal disciplinado. Se o programa é tão simples que um plano parece insensato, o plano deve ser trivial de produzir.

4.2.2 O processo

Um *script* guia o estudante de PSP através de um processo. Os elementos principais deste *script* são seu propósito, os critérios de entrada, as fases (ou passos) a serem executadas, e os critérios de saída. O *Script* do Processo PSP0 é mostrado na Tabela A-1, no apêndice A. Ele descreve em palavras a estrutura de processo mostrada na Fig. 4-3. Um segundo *script* do PSP0, o *Script* de Planejamento, é mostrado na Tabela A-2. Ele resume brevemente os passos simples de planejamento requeridos no PSP0. As fases de planejamento e *Postmortem* estão bastante claras nos *scripts* das Tabelas A-2 e A-4, mas a fase de desenvolvimento na Tabela A-3 tem quatro passos: projeto, codificação, compilação e teste. Até que estes passos tenham critérios explícitos de entrada e saída, não há nenhuma maneira de dizer quando começa ou termina cada um. Uma confusão comum nesses casos diz respeito à distinção entre codificação e compilação.

Medidas do PSP0

O PSP0 tem duas medidas:

- o tempo gasto por fase: registro do tempo que se gasta em cada parte de processo do PSP. O objetivo é determinar onde o tempo é gasto e como esta distribuição muda à medida que o processo muda.
- os defeitos por fase: registro dos dados de cada defeito (mudança do programa) localizado durante a compilação e testes.

Junta-se dados de tempo e defeitos para obter a base para planejar os projetos futuros. Eles dão uma base contra a qual medir o desempenho, mostram onde se passa mais tempo e indicam onde se faz e se acha a maioria dos defeitos. Eles também ajudarão a ver como estas distribuições mudam à medida que o processo evolui.

A Tabela A-7 mostra o Log de Registro de Tempo, e a Tabela A-8 contém as instruções para completá-lo.

O Log de Registro de Defeito e suas instruções são mostradas nas Tabelas A-9 e A-10. Quando o estudante[9] terminar de corrigir o defeito, deve anotar o Tipo de Defeito Padrão mostrado na Tabela A-11. Este padrão foi modelado na IBM Research, e deve ser suficientemente geral para cobrir a maioria das necessidades.

O Sumário de Plano de Projeto do PSP0 e suas instruções são mostrados nas Tabelas A-5 e A-6.

Embora seja possível ajustar o modo particular de cada um de projetar *software*, Humphrey sugere que os processos do PSP não sejam modificados até a conclusão do livro, ou curso de PSP, principalmente pelo grande trabalho extra que isso acarretaria.

4.2.3 PSP0 - Conteúdos do processo

O PSP0 apresenta a família de processos do PSP e seus formulários, *scripts*, e padrões. Provê uma estrutura ordenada para planejar o trabalho e relatar os resultados, estes elementos de processo podem economizar uma quantia significativa de tempo. Sem formulários padrão, por exemplo, haveria a necessidade de decidir como produzir um plano, quais dados juntar, e como registra-los.

Como mostrado no *script* da Tabela A-1, é necessário,

1. assegurar-se que se têm todas as entradas requeridas,
2. assegurar-se que os requerimentos para o trabalho foram entendidos,
3. calcular o tempo em minutos que se espera levar para o desenvolvimento do programa,
4. registrar esse tempo no Resumo de Plano de Projeto, inclusive o tempo que o planejamento tomou,
5. fazer o desenvolvimento,
6. entrar com os dados atuais dos Logs de Registro de Tempo e Defeito na coluna Atual do Sumário de Plano de Projeto.

Ao fazer o desenvolvimento do programa, o projeto, a implementação, a compilação e os testes são efetuados usando os métodos de desenvolvimento atuais do engenheiro de *software*. Também, são registrados os defeitos no Log de Registro de Defeito e o tempo no Log de Registro de Tempo. Cada fase de processo é descrita nos *Scripts* de Planejamento, Desenvolvimento e *Postmortem*.

[9] Estudante do PSP.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Os objetivos do PSP0 são:

- incorporar medidas básicas no processo de desenvolvimento de *software*,
- requerer mudanças mínimas nas práticas pessoais,
- demonstrar o uso de um processo definido ao se escrever programas pequenos, e
- usar o processo atual como uma estrutura de processo introdutório.

As condições prévias para o PSP0 são que o desenvolvedor seja razoavelmente fluente em pelo menos uma linguagem de programação. O processo PSP0 e seus *scripts*, formulários, modelos[10], padrões e instruções são descritos em termos gerais nas seções anteriores. É necessário revisar esses elementos antes de começar a escrever o primeiro programa.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

Os *scripts*, formulários, modelos e padrões são os itens usados no processo PSP0. Os *scripts* e o Resumo de Plano de Projeto mudam em cada processo, mas o Log de Registro de Tempo, o Log de Registro de Defeito e o Tipo de Defeito Padrão são usados sem mudanças em todas as versões de processos subseqüentes. Todos estes itens são incluídos nos números de tabela indicados.

PSP0 - *Script* de Processo	**Tabela** A-1
PSP0 - *Script* de Planejamento	**Tabela** A-2
PSP0 - *Script* de Desenvolvimento	**Tabela** A-3
PSP0 - *Script* de *Postmortem*	**Tabela** A-4
PSP0 - Sumário do Plano de Projeto e Instruções	**Tabela**s A-5 e A-6
Log de Registro de Tempo e Instruções	**Tabela**s A-7 e A-8
Log de Registro de Defeito e Instruções	**Tabela**s A-9 e A-10
Tipo de Defeito Padrão	**Tabela** A-11

Os Elementos do PSP0

Como o PSP0 é o primeiro processo do PSP, todos os seus elementos são novos. Os *scripts*, Log de Registro de Tempo, Log de Registro de Defeito e Tipo de Defeito Padrão foram descritos no início desta seção.

ESPECIFICAÇÃO DE RELATÓRIO DE PROCESSO

O objetivo principal dos exercícios do PSP0 é proporcionar experiência no uso de um processo disciplinado. Embora seja importante produzir um programa funcionando, os critérios para avaliar os exercícios do PSP0 são:

---

[10] Modelos (*templates*): lidam com um volume de dados variáveis. Formulários (*forms*): tratam de uma quantidade fixa de dados.

- os dados de processo estão completos;
- os dados são precisos e autoconsistentes;
- o relatório de processo é submetido na própria ordem e formato.

O uso de um formato padrão e ordenado torna mais fácil para o estudante assegurar-se de que os resultados estão completos e para o instrutor[11] assegurar-se que eles estão corretos. Os itens a serem incluídos nos relatórios dos exercícios do PSP0 e a **ordem** nas quais eles serão submetidos são como segue:

- Resumo do Plano de Projeto do PSP0
- Log de Registro de Tempo
- Log de Registro de Defeito
- Listagem do programa fonte

4.2.4 Exercícios

A tarefa padrão para este capítulo usa o PSP0 para escrever o programa 1A. Completando esta tarefa, deve-se seguir fielmente o formato de submissão de relatório especificado para o PSP0.

Programa 1A

Programa 1A - Requerimentos*:* Escrever um programa para calcular o ponto central e o desvio padrão de uma série de números reais n. O ponto central é a média dos números. A fórmula para o desvio padrão é:

$$s(x_1, \ldots, x_n) = \sqrt{\left(\left(\sum_{i=1}^{n} (x_i - x_{avg})^2 \right) / (n-1) \right)}$$

Figura 4-4 - Desvio Padrão

onde b(x1,… ,x) é a desvio padrão do valor x e x_{avg} é a média destes n valores. Usar uma lista encadeada para guardar os números **n** para os cálculos.

Programa 1A - Teste: testar o programa completamente. Pelo menos três dos testes devem usar os dados de cada uma das três colunas da Tabela 4-1. Os desvios padrão para as colunas nesta tabela são LOC Objeto: 572,03; LOC Novas e Mudadas: 625,63 e Horas de Desenvolvimento: 62,26.

[11] O Livro *A Discipline For Software Engineering* foi projetado para ser usado em cursos de PSP, podendo ser usado de forma autodidata.

Tabela 4-1 - Tamanho do Objeto Pascal e Horas de Desenvolvimento

TAMANHO DO OBJETO PASCAL E HORAS DE DESENVOLVIMENTO			
Programa Número	LOC Objeto	LOC novas e mudadas	Horas de desenvolvimento
1	160	186	15.0
2	591	699	69.9
3	114	132	6.5
4	229	272	22.4
5	230	291	28.4
6	270	331	65.9
7	128	199	19.4
8	1657	1890	198.7
9	624	788	38.8
10	1503	1601	138.2

4.3 PSP0.1 – Medindo o Tamanho do *Software*

4.3.1 O Processo de Planejamento

O planejamento é o primeiro passo do PSP por três razões. Primeiro, sem bons planos não é possível administrar efetivamente nem mesmo projetos de *software* de tamanho modesto. Segundo, planejar é uma habilidade que se pode aprender e melhorar com a prática. Terceiro, boas habilidades de planejamento ajudarão a fazer um melhor trabalho de *software*.

Embora o planejamento pessoal seja importante para o planejamento de projetos, é só uma parte do processo. Muitos outros assuntos estão envolvidos na produção de um plano completo para um grande projeto. Porém, estes planos de projeto maiores serão provavelmente mais realísticos quando eles estiverem compostos de planos pessoais múltiplos feitos pelos indivíduos ou grupos que farão o trabalho.

4.3.1.1 O que é um Plano

O plano de projeto define o trabalho e como será feito. Provê uma definição de cada tarefa principal, uma estimativa do tempo e recursos requeridos e uma estrutura para revisão da administração e controle. O plano de projeto também é um poderoso veículo de aprendizado. Quando corretamente documentado, é um ponto de referência

para comparar com o desempenho atual. Esta comparação permite aos planejadores verem seus erros estimados e melhorar a precisão de estimativas [HUM 89, *apud* [HUM 95]] Assim, um plano é muitas coisas. Em organizações de *software* maduras, planos são tipicamente usados como:

- uma base para concordar sobre custos e horários para um trabalho,
- uma estrutura organizada para fazer o trabalho,
- uma estrutura para obter os recursos requeridos, e
- um registro do que foi inicialmente comprometido.

Os planos do PSP têm dois usuários: o **desenvolvedor** e seus **clientes**. Para o trabalho do desenvolvedor, são necessárias quatro coisas gerais para um plano:

- medir o tamanho do trabalho: qual o tamanho do trabalho, quanto tempo ele levará?
- estrutura de trabalho: como será feito o trabalho? O que será feito primeiro, segundo, e assim por diante?
- estado do trabalho: como se sabe onde se está? O desenvolvedor vai terminar na hora certa e os custos estão sob controle?
- avaliação: qual a qualidade do plano? Houve algum erro óbvio, que enganos deveriam ser evitados no futuro, e como se pode fazer um trabalho melhor da próxima vez?

Os clientes poderiam ser o instrutor do curso de PSP, os colegas de trabalho, o gerente ou um usuário final. Estas pessoas também querem quatro coisas gerais do plano:

- qual é o compromisso? Especificamente, o que será entregue, quando e a que custo?
- qual será a qualidade deste produto? Será a que eles quiseram? O trabalho está corretamente planejado para assegurar os ajustes do produto às suas necessidades? Há provisões para eles fazerem checagens na qualidade de produto? Há provisões para solucionar assuntos?
- há algum modo para monitorar o progresso? Eles terão cedo advertência de custos, horários ou problemas de qualidade? Nesse caso, quando eles podem descobrir os problemas e o que podem fazer sobre eles?
- eles serão capazes de avaliar a qualidade do trabalho terminado? Eles podem separar os problemas causados pelo planejamento ruim daqueles causados por mau gerenciamento? O impacto de mudanças de escopo estará claro e identificável?

Quando se examina o plano no contexto destas perguntas, várias coisas ficam claras:

1. O plano deve estar baseado em se fazer uma parte definida do trabalho.
2. O trabalho deve envolver passos múltiplos que estão claramente definidos e medidos. Isto provê uma estrutura para o plano e uma base para monitorar o progresso.
3. Será necessário algum modo de conferir o plano com o usuário antes de se começar o trabalho. Esta sempre é uma boa idéia, e é essencial para qualquer trabalho, mesmo para as menores tarefas.

4. Será necessário fazer declarações de progresso periódicas aos clientes.

4.3.1.2 Planejando um Projeto de *Software*

Os passos seguintes ajudarão o desenvolvedor a construir uma estimativa de processo estável e efetiva:

- Começar com uma declaração explícita do trabalho a ser feito e conferir para se assegurar que é o que o cliente espera (os modos nos quais os projetos podem diferir são infinitos).
- Para projetos que levam mais do que alguns dias de trabalho, dividí-los em múltiplas tarefas menores e calcular cada tarefa separadamente. O detalhe adicionado vai melhorar a precisão do plano e provavelmente melhorará a acurácia do desenvolvedor.
- Basear as estimativas, comparando o trabalho atual com os dados históricos dos trabalhos anteriores.
- Registrar as estimativas e depois compará-las com os resultados atuais.

4.3.1.3 Estrutura de planejamento

A estrutura de planejamento do PSP é mostrada na Fig. 4-5. As tarefas executadas são mostradas nos retângulos e os vários dados, relatórios e produtos são mostrados nos ovais. Aqui, começando com uma necessidade do cliente, são definidas as exigências (requerimentos). A seguir, é produzido um projeto conceitual que em troca ajuda a relacionar a estimativa de planejamento ao produto atual que se pretenda construir.

Com este projeto conceitual e dados históricos de produtos previamente construídos, é possível estimar o tamanho provável do novo produto. Com esta estimativa de tamanho, usa-se os dados históricos de produtividade para estimar quantas horas o trabalho levará. Finalmente, estas horas são alocadas em um calendário para se ter um horário de projeto. Com estes dados e com uma data presumida de início, é possível estimar agora a data que o trabalho terminará.

Com o plano em mãos, e assumindo que se tenha toda a informação e facilidades necessárias, o desenvolvimento é iniciado. Durante o desenvolvimento e a conclusão do projeto, são registrados o tempo gasto e o tamanho do produto produzido. Estes dados são usados para produzir relatórios periódicos e fazer análises de processo. Estas análises fornecem os dados de tamanho e produtividade para fazer planos futuros.

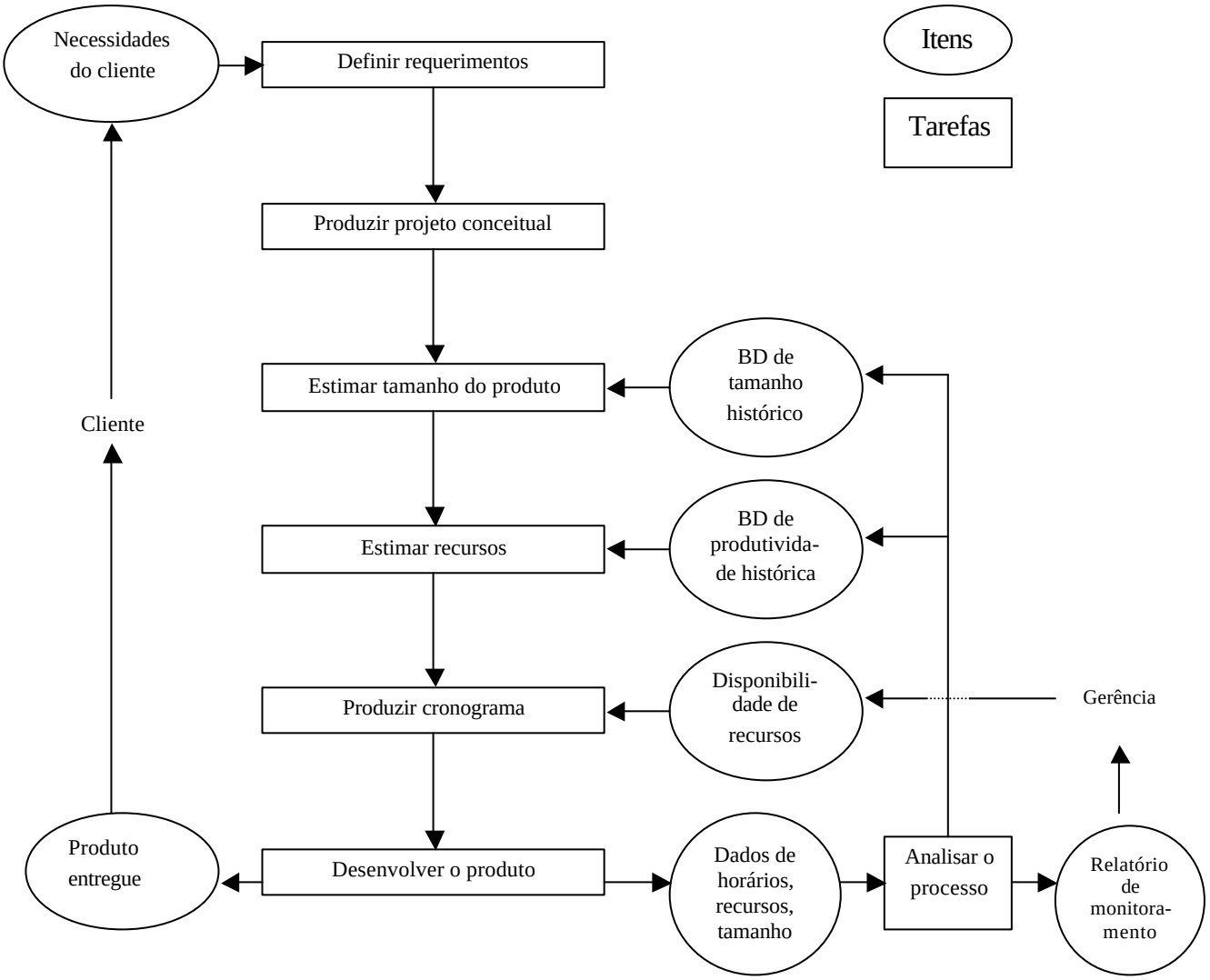

Figura 4-5 - Estrutura de Planejamento de Projeto

4.3.1.4 Produzindo um Plano de Qualidade

À medida que engenheiro de *software* constrói suas habilidades de planejamento, ele precisa pensar na qualidade do plano. O que constitui um bom plano? Qual a precisão desses planos e o que ele pode fazer para melhorar suas habilidades de planejamento? As seis questões chaves do plano são as seguintes:

1. Está completo?
2. É acessível?
3. Está claro?
4. É específico?

5. É preciso?
6. É acurado?

4.3.2 Medindo o tamanho do *software*

O processo de planejamento de *software* começa com uma estimativa de tamanho do trabalho. Antes de poder estimar o tamanho do *software*, porém, é necessário um modo consistente e repetível para descrever o tamanho de um produto.

O Projeto de Medida de Processo de *Software* do SEI desenvolveu um trabalho para descrever medidas de tamanho de *software*. Os dois critérios principais para este trabalho foram os seguintes:

- Comunicação: "Se alguém usar nossos métodos para definir uma medida ou descrever um resultado de medida, outros saberão precisamente o que foi medido e o que foi incluído e excluído?"
- Repetibilidade: "Outra pessoa poderia repetir a medida e receber o mesmo resultado?"

Conhecendo estes objetivos, o SEI estabeleceu uma estrutura para definir precisamente a métrica de LOCs. Uma versão simplificada do formulário que eles desenvolveram é mostrada na Tabela A-12.

Considerando que a contagem de LOC pode estar enganada, ela deve ser tratada com um pouco de cuidado. Deve-se usar definições precisas e anotar cuidadosamente os tipos de elementos de programa incluídos. Não é uma boa idéia usar contagem de LOC para comparar projetos ou organizações porque geralmente há muitas diferenças que não são detectáveis de simples dados de LOCs. Até mesmo ao nível do PSP, a contagem de LOCs não é uma base útil para se comparar a produtividade ou efetividade de indivíduos.

Ao medir a produtividade do desenvolvimento, os engenheiros de *software* normalmente contam o número de declarações fonte por hora de desenvolvimento. Para este propósito, deve-se contar os **recentes desenvolvimentos mais as declarações modificadas**. Também é importante usar precisamente as mesmas definições ao se estimar a produtividade de desenvolvimento e no planejamento de projeto.

Quando compara a taxa de defeitos de programas, o engenheiro, comumente, usa a taxa de defeitos por mil linhas de código adicionado e modificado. Ao estimar a carga provável de trabalho de manutenção para um programa, porém, será mais apropriado considerar as LOCs totais do produto terminado.

Um contador de LOCs pode ser projetado para contar linhas físicas ou linhas lógicas. Um terceiro método combina estes dois: conta linhas lógicas usando um padrão de codificação e um contador de LOCs físicas. Esta é a abordagem usada com o PSP.

Podem ser obtidas muitas estatísticas de LOCs úteis com ferramentas corretamente projetadas. Para cada modificação do programa, por exemplo, o engenheiro poderá querer saber quantas LOCs foram adicionadas e apagadas. Um modo prático de obter tais dados é com um programa que compare cada versão do programa com a versão anterior. Este comparador então identifica e conta cada linha adicionada ou apagada.

A chave para medidas de tamanho de *software* efetivas é assegurar que elas se ajustem às necessidades pessoais do desenvolvedor.

4.3.3 PSP0.1 - Conteúdos do processo

Ao PSP0 é acrescentado o PSP0.1, adicionando um padrão de codificação, medidas de tamanho e a proposta de melhoria de processo (PIP - *Process Improvement Proposal*). A PIP é um formulário que provê um modo estruturado para registrar problemas de processo, experiências e sugestões de melhorias.

Antes de usar o PSP0.1 para desenvolver um programa, faz-se um cálculo do tamanho e uma estimativa de tempo completa. Na conclusão do desenvolvimento, mede-se o tamanho do programa completado e conta-se ou calcula-se as LOCs reusadas, apagadas, modificadas, adicionadas, totais novas e alteradas e totais novas reusadas.

Objetivos e Condições Prévias

Em adição aos objetivos do PSP0, este processo tem o objetivo adicional de ajudar a medir e calcular os tamanhos dos programas produzidos. As condições prévias para o PSP0.1 são a presente seção, o PSP0 e os padrões de codificação e contagem de LOCs produzidos com as tarefas R1 e R2, mais adiante explicitados.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP0.1 - *Script* de Processo	**Tabela A-31**
PSP0.1 - *Script* de Planejamento	**Tabela A-32**
PSP0.1 - *Script* de Desenvolvimento	**Tabela A-33**
PSP0. 1 - *Script* de *Postmortem*	**Tabela A-34**
PSP0.1 - Resumo de Plano de Projeto e Instruções	**Tabelas A-35 e A-36**
Proposta de Melhoria de Processo (PIP) e Instruções	**Tabelas A-37 e A-38**
Padrão de codificação	**Tabela A-13**
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ELEMENTOS DE PROCESSO NOVOS

Os elementos novos para o PSP0.1 não só incluem *scripts* e o relatório sumário mas também a proposta de melhoria de processo (PIP) e padrões de codificação. As seções seguintes dão uma descrição desses elementos.

Proposta de Melhoria de Processo. O formulário PIP é usado para registrar qualquer problema ou sugestões de melhoria que ocorram no uso de um processo. Muitos problemas de definição de processo concernem a detalhes aparentemente secundários. Porém, são tão detalhados que fazem a diferença entre um processo aborrecido e inconveniente e um processo confortável e eficiente. Embora seja possível lembrar-se posteriormente que algum formulário ou passo foi um problema, provavelmente os detalhes serão esquecidos se não forem registrados prontamente. O formulário PIP em branco deve ser mantido sempre à mão para, prontamente, registrar-se qualquer idéia de melhoria. Detalhes são importantes em um processo pessoal e, para fixar os detalhes, é necessário completar os PIPs regularmente.

Os PIPs devem ser usados também para registrar comentários ou fazer notas sobre o que foi aprendido em cada programa. Ao se fazer as tarefas, os PIPs devem ser submetidos com os relatórios de processo.

O formulário PIP é mostrado na Tabela A-37 e suas instruções estão na Tabela A-38. Deve-se reter uma cópia de todos os PIPs completados, para uso posterior.

Padrões de codificação. Além dos programas estarem corretos, o código fonte também deve ser compreensível. Escrever código legível ajudará a pensar mais claramente no projeto e nos testes, modificações e reusos. Comentários claros e precisos também ajudam outras pessoas que trabalham com esses programas.

O padrão de codificação do PSP, para C++, é mostrado na Tabela A-13.

Para ser útil, o padrão de codificação deve ser projetado para a linguagem e o ambiente de trabalho em uso. Assim, é possível alterar o padrão de codificação do PSP para as necessidades particulares, mas deve-se reter todos os itens que este padrão inclui.

Especificações de Relatório de Processo

Os critérios para avaliar os exercícios do PSP0.1 neste texto são iguais aos exercícios do PSP0:

- Os dados de processo estão completos.
- Os dados são precisos e autoconsistentes.
- O relatório de processo é submetido na própria ordem e formato.

Os itens a serem incluídos nos relatórios de exercícios do PSP0.1 e a ordem nas quais eles serão submetidos são como segue:

- **PSP0.1 - Resumo de Plano de projeto.**
- **O formulário PIP, inclusive com uma declaração breve de lições aprendidas.**
- Log de Registro de Tempo.
- Log de Registro de Defeito.
- Listagem do programa fonte.

4.3.4 Exercícios

A tarefa padrão para esta seção inclui o exercício R1 e R2 e usa o PSP0.1 para escrever o programa 2A. Ao completar esta tarefa, deve-se rever e familiarizar-se com o processo PSP0.1.

R1 - PADRÃO DE CONTAGEM DE LINHAS DE CÓDIGO

Avaliação da tarefa: produzir uma especificação padrão de contagem de LOCs.

Objetivos de tarefa:

- Definir os padrões de contagem de LOCs que sejam apropriados para a linguagem de programação usada;
- Prover uma base para desenvolver um padrão de codificação;
- Preparar-se para desenvolver um programa para contar LOCs.

Tarefa: Exercício R1 - Desenvolver e submeter um padrão para contagem de LOCs lógicas para a linguagem de programação usada para escrever os programas dos exercícios do PSP.

Resultado: produzir, documentar e submeter o padrão completo, usando o formato da Tabela A-12 no Apêndice A.

Fazer esta tarefa antes ou junto com a escrita do programa 2A.

R2 - PADRÃO DE CODIFICAÇÃO

Avaliação da tarefa: produzir um padrão de codificação para uso na escrita dos programas de exercícios do PSP. Objetivos de tarefa:

- Estabelecer um conjunto consistente de práticas de codificação;
- Prover critérios para julgamento da qualidade do código produzido;

- Facilitar a contagem de LOCs, assegurando que os programas são escritos com uma linha física separada para cada linha lógica de código.

Tarefa: Exercício R2 - produzir um padrão de codificação que exija qualidade de práticas de codificação. Usar o padrão de codificação do PSP da Tabela A-13. Também assegurar-se que uma linha física separada é usada para cada linha lógica de código, como definido no exercício R1.

Resultado: produzir, documentar e submeter o padrão de codificação completo.

Cronograma: fazer esta tarefa depois de produzir o relatório R1, antes ou junto com a escrita do programa 2A.

Programa 2A

Condições prévias e referências: seção atual e exercícios R1 e R2.

Programa 2A: escrever um programa para contar as linhas lógicas em um programa, omitindo comentários e linhas em branco. Usar o contador padrão produzido através do exercício de relatório R1 para colocar uma linha lógica em cada linha física e contar linhas físicas. Produzir uma única contagem para o arquivo fonte do programa inteiro.

Testar completamente o programa. Como teste, contar as LOCs nos programas 1A e 2A. Submeter estes dados usando o formato da Tabela A-14.

4.4 PSP1 – Estimando tamanho e tempo

4.4.1 Estimando o tamanho do *software*

Esta seção discute primeiro o problema da estimativa de tamanho e então descreve o método de estimativa PROBE usado no livro.

A razão principal para estimar o tamanho de um produto de *software* é ajudar a planejar o desenvolvimento do produto. A qualidade de um plano de desenvolvimento de *software* geralmente depende da qualidade da estimativa de tamanho. Um bom plano fornece a base para consolidar e prover de pessoal o trabalho, para saber o que tem que ser feito, quando e por quem. Uma causa freqüente de planos ruins é uma estimativa de tamanho ruim.

Uma prática aceita na engenharia, na manufatura e na construção civil é basear planos de desenvolvimento em estimativas de tamanho de produto. Engenheiros civis experientes podem freqüentemente calcular grandes projetos com uma margem de erro de um ou dois por cento do custo total atual.

O grau para o qual se pode acurada e precisamente planejar um trabalho depende do que se sabe sobre ele. No início, ou no estágio de pré-proposta, só se tem uma idéia geral dos requerimentos do produto. Praticamente a única maneira de fazer uma estimativa é por analogia a produtos prévios.

Depois da fase de pré-proposta, se sabe progressivamente mais sobre o produto planejado. Pode-se, então, fazer estimativas mais refinadas de tamanho do trabalho.

Estimativas para trabalhos de *software* grandes podem ser controladas desse modo. Para fazer uma estimativa precisa, começa-se com uma especificação de projeto. Então cada parte do trabalho é examinada e estimada. Esta estimativa requer estimativas separadas para cada componente de *software*, cada documento principal, os casos de teste, planejamento da instalação, conversão de arquivos e treinamento de usuários. Os componentes de programa podem ter diferentes sub-elementos. Se há telas para desenvolver, relatórios para gerar ou lógica funcional para projetar, estes também devem ser calculados. Para grandes produtos de *software*, há potencialmente muitos detalhes úteis. Para fazer estimativas precisas, é necessário definir todos estes detalhes e considerar cada um na estimativa.

4.4.1.1 Critérios de estimativa de tamanho

Um método de estimativa de tamanho utilizável deve satisfazer os seguintes critérios:

- deve usar métodos estruturados e treináveis. Um método estruturado facilita a melhoria do treinamento e processo. Também permite monitorar e melhorar o próprio método de estimativa.
- deve ser um método que se possa usar durante todas as fases de desenvolvimento de *software* e manutenção.
- deve ser utilizável por todos os elementos do *software*. Não só deve controlar o código, mas também arquivos, relatórios, telas e documentação.
- deve ser satisfatório para análise estatística. Um método de estimativa de tamanho estatisticamente baseado provê os meios para ajustar as estimativas de parâmetros baseadas em dados históricos.
- deve ser adaptável aos prováveis tipos de trabalho a serem feitos no futuro. À medida que se construa dados de estimativa e se ganhe experiência, se estará então construindo um recurso de valor contínuo.
- deve prover os meios para julgar a precisão das estimativas.

Deve-se sempre comparar cada estimativa com o tamanho do produto resultante atual. Revisando estas comparações, freqüentemente as causas dos erros serão conhecidas e o método de estimativa poderá ser melhor ajustado e melhorado.

Métodos de estimativa populares

Boehm descreveu o método ***Wideband-Delphi*** para opiniões múltiplas de *experts*. Putnam definiu o método de **lógica *fuzzy*** e o método de **componente-padrão** para julgar os tamanhos de produtos novos baseado em dados de tamanho históricos. O método de **ponto por função** de Albrecht usa fatores *standards* para julgar a importância relativa de vários requerimentos funcionais. Estes métodos formam a fundação para o método PROBE usado com o PSP.

4.4.1.2 *Proxy Based Estimating* - PROBE

No planejamento de projetos, geralmente são requeridas estimativas antes do desenvolvimento começar. Nesta fase inicial, os requerimentos podem ser entendidos mas pouco é conhecido sobre o próprio produto. O problema da estimativa é, assim, predizer o tamanho provável terminado do produto. Como ninguém pode saber com antecedência o tamanho que um produto planejado terá, a estimativa de tamanho de *software* sempre será um processo incerto. Porém, a necessidade é fazer uma estimativa tão acurada quanto possível. Em geral, todos os métodos de estimativa usam dados de programas semelhantes previamente desenvolvidos para estabelecer alguma base para julgar o tamanho do novo programa.

4.4.1.3 Proxies

Exemplos de *proxies* são objetos, telas, arquivos, *scripts* ou pontos de função. Os critérios para uma boa *proxy* são como segue:

- A medida de tamanho da *proxy* deve se relacionar de perto ao esforço exigido para desenvolver o produto.
- O conteúdo da *proxy* de um produto deve ser automaticamente contável.
- A *proxy* deve ser fácil de visualizar no começo de um projeto.
- A *proxy* deve ser customizável às necessidades especiais das organizações.
- A *proxy* deve ser sensível a qualquer variação de implementação que impactem custos de desenvolvimento ou esforços.

Muitos tipos potenciais de *proxies* poderiam satisfazer os critérios previamente esboçados. O método de ponto por função é um candidato óbvio porque é extensamente usado. Muitas pessoas acham o método ponto por função útil para o recurso de estimativas. Outras possíveis *proxies* são objetos, telas, arquivos, *scripts* e capítulos de documentos.

4.4.2 O método PROBE (*Proxy-Based Estimating*)

O PSP1 formaliza a estimativa de tamanho do programa. Introduz o PROBE, que é um método algorítmico para a estimativa de tamanho. O PROBE analisa o banco de dados histórico e olha para uma tendência na relação entre tamanho calculado e

atual. Assume que o desempenho do passado é indicativo do desempenho futuro. Se os bancos de dados históricos mostram que o tamanho do projeto final é tradicionalmente mais alto que a estimativa original do programador por alguma porcentagem, o PROBE permite ao usuário aumentar a estimativa original dele por aquela porcentagem. Usa uma análise de regressão linear na relação entre o tamanho calculado e o tamanho atual. A equação da linha de melhor ajuste é usada para converter uma estimativa inicial em uma estimativa projetada. Pelo uso da distribuição *t-Student*, um salto no erro de estimativa pode ser colocado para um determinado nível de confiança. Se a relação linear entre os pontos de dados for fraca, o PROBE proporcionará para um resultado uma alta gama de erro. [COU 98]

Um fluxograma do procedimento de estimativa de tamanho PROBE é mostrado na Fig. 4-6. Estes passos também são explicados no ***script* PROBE** na Tabela A-45 e nas instruções do modelo de estimativa de tamanho (Tabela A-49).

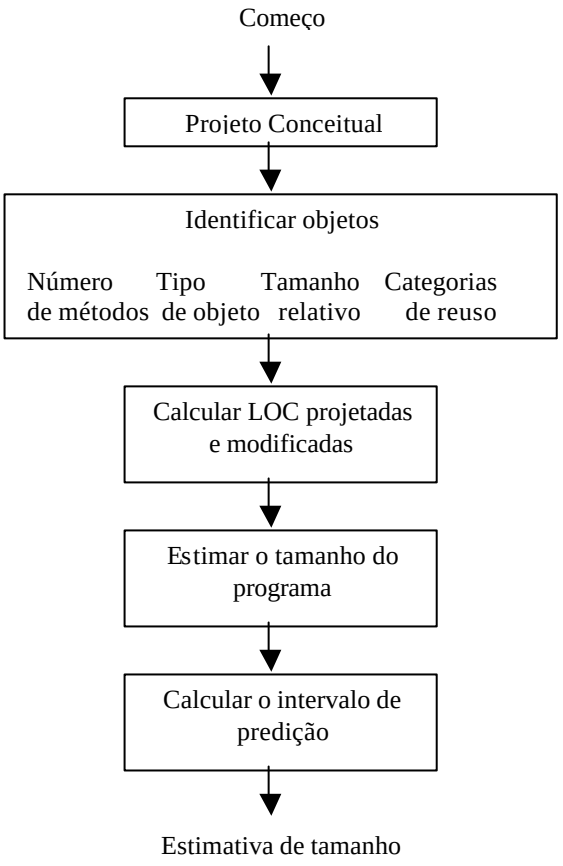

Figura 4-6 - O método *Proxy-Based Estimating* (PROBE)

O projeto conceitual

Para a estimativa de tamanho refletir corretamente o produto a ser construído, é necessário começar com um projeto conceitual. Este projeto estabelece uma abordagem

de projeto preliminar e nomeia os objetos esperados e suas funções. Sua intenção aqui não é fazer o projeto completo mas postular os objetos que serão necessários e as funções que eles executarão. Este é um processo de abstração durante o qual o estimador diz, "Se eu tivesse objetos que fizessem as funções A, B, e C, eu saberia construir este produto".

Ao estimar produtos relativamente pequenos, é possível produzir um projeto conceitual diretamente. No projeto conceitual de grandes produtos, o objetivo é dividir cada uma das partes principais do produto em elementos que se assemelham a aqueles com os quais o desenvolvedor tem experiência e dados. Se um objeto está no nível certo e não pertence a quaisquer das categorias existentes, então seu tamanho é estimado como o primeiro de uma nova categoria.

Determinação do tipo de objeto e do tamanho

Quando já se tem o projeto conceitual, cada objeto foi nomeado e sua categoria já foi determinada, é necessário localizar objetos no banco de dados que se assemelhem a cada um destes objetos. Para cada novo objeto, é feito um julgamento de como seu tamanho se compara com os do banco de dados em sua categoria. Com base nesse julgamento, é feita uma estimativa superficial do tamanho do novo objeto.

Por exemplo, considerar a estimativa que um estudante do PSP[12] fez para o programa l0A em C++, mostrada no formulário de estimativa na Tabela 4-2, abaixo.

Tabela 4-2 - Exemplo de Estimativa de Tamanho

Student	**Estudante 12**		**Date**	**05/01/94**	
Instructor	**Humphrey**		**Program #**	**10A**	
BASE PROGRAM				ESTIMATE	ATUAL
BASE SIZE (B) => => => => => => => => => =>				695	695
LOC DELETED (D) => => => => => => => => =>				0	0
LOC MODIFIED (M) => => => => => => => => =>				5	18
PROJECTED LOC					
BASE ADDITIONS:	TYPE	METHODS	REL. SIZE	LOC	LOC
TOTAL BASE ADDITIONS (BA) => => => => => =>					
NEW OBJECTS:	TYPE[1]	METHODS	REL. SIZE	LOC (NewReuse*)	
Matrix	Date	13	Medium	115	136
Linear System	Calc.	8	Large	197	226

[12] Estimativa do estudante 12 do livro de Humprey [HUM 95]

Linked List	Data	2	Large	49*	54
Control	Logic	2			48
TOTAL NEW OBJECTS (NO) => => => => => => =>				361	464
REUSED OBJECTS				LOC	LOC
Linked List				73	73
Data Entry				96	96
REUSED TOTAL (R) => => => => => => => => =>				169	169
Estimated Object LOC (E)		E = BA+NO+M		366	
Regression Parameter:		$ß_{0\ (size\ and\ time)}$		62,0	108,0
Regression Parameter:		$ß_{1\ (size\ and\ time)}$		1,3	2,95
Estimated New and Changed LOC:		$N = ß_0 + ß_1 *(E)$		538	
Estimated Total LOC:		T = N + B - D - M + R		1397	
Estimated Total New Reused (sum of * LOC):				49	
Estimated Total Development Time:		Time = ß0 + ß1 *(E)			1186
Prediction Range:		Range		235	431
Upper Prediction Interval:		UPI = N + Range		773	1617
Lower Prediction Interval:		LPI = N - Range		303	755
Prediction Interval Percent:				90%	90%

[1] L-Logic, I-I/O, C-Calculation, T-Text, D-Data, S-Set-up

Usando o projeto conceitual, o estudante nomeou cada objeto novo e determinou o seu tipo. O primeiro objeto, Matrix, era do tipo de Dados. O estudante a seguir estimou quantos métodos ou procedimentos este objeto provavelmente conteria, neste exemplo, 13. A seguir, o estudante julgou o tamanho relativo deste objeto como médio e determinou do seu banco de dados histórico para objetos C++ que um objeto de dados médio teria 8.84 LOC por método. Multiplicando o número de métodos, resultou um 114.9 LOC. O estudante, então, repetiu este procedimento para cada objeto novo para dar um total de 361 LOC de objeto novas.

Programa base

Ao mesmo tempo em que se está estimando os objetos novos, também se determinam o tamanho do programa básico que se está aumentando e qualquer mudança dele. A Tabela 4-2 mostra que o Estudante 12 identificou 695 LOC de código básico, 5 das quais ele planejou modificar.

Objetos reusados

Se for possível achar objetos ou procedimentos disponíveis que poderiam prover as funções requeridas pelo projeto conceitual, é possível reusá-los. No exemplo da Tabela 4-2, o estudante 12 identificou dois objetos reusados com um total de 169 LOC.

O método PROBE considera dois tipos de reusados. Os primeiros são os objetos reusados tomados da biblioteca de reuso. Os segundos, os objetos reusados novos, são objetos novos que se planeja desenvolver. São identificados como objetos reusados quando são suficientemente gerais para serem postos dentro da biblioteca de reuso.

Cálculo de LOC objetos estimadas

Começando no topo do exemplo da Tabela 4-2, entrar com os vários totais. A base (B) é igual a 695 LOC, apagadas (D) é 0, e modificadas (M) é 5. Adições Básicas (BA) é igual a 0. Há três Objetos Novos (NO) daquele total de 361 LOC, e 49 LOC destas são novas reusadas. Assim E = 0 + 361 + 5 = 366.

Regressão linear

Uma vez que se tenham as LOC objeto, é necessário um modo para calcular o tamanho total do programa. Um modo simples de fazer isto é olhar para o histórico de desenvolvimento pessoal. Se, por exemplo, os dados históricos dos programas acabados sempre são 25 por cento maiores que o total estimado, seria somado 25 por cento para se obter a estimativa para o programa finalizado. Tudo o que os cálculos de regressão linear fazem é calcular uma fórmula para fazer esta conversão. A fórmula de regressão linear para fazer este cálculo é:

Tamanho de programa = ß0 + LOC Objeto estimadas * ß1

Ou, mais geralmente,

$$y_k = ß\,0 + x_k\, ß\,1$$

Figura 4-7 - Tamanho do programa

Os parâmetros de estimativa ß0 e ß1 são calculados dos dados históricos usando as seguintes equações:

$$ß1 = \left(\sum_{i=1}^{n} x_i y_i - n x_{avg} y_{avg}\right) / \left(\sum_{i=1}^{n} x_i^2 - n(x_{avg})^2\right)$$

Figura 4-8 - Parâmetro de Estimativa ß1

$$ß0 = y_{avg} - ß1x_{avg}$$

Figura 4-9 - Parâmetro de Estimativa ß0

Por exemplo, aqui x_1 seria o tamanho de objeto estimado originalmente para o programa 1 e y_1 seria o tamanho do programa terminado total do programa 1. Semelhantemente, x_2 e y_2 seriam os dados para o programa 2, e assim sucessivamente. Também, x_{avg} é a média de todos os termos x_i e y_{avg} é a média de todos os termos y_i.

Determinando o tamanho estimado do programa

O produto acabado conterá mais que os objetos há pouco calculados. Por exemplo, as estimativas de objeto não incluem a rotina principal ou declarações de código de cabeçalho. Para estes, aplica-se algum fator baseado na experiência histórica do desenvolvedor. O *script* de estimativa PROBE, Tabela A-45, descreve como obter os parâmetros de regressão linear ß0 e ß1 para fazer isto.

Intervalo de predição

Uma vez feita uma estimativa, é necessário avaliar a sua qualidade. Usando dados históricos e algo chamado t, ou distribuição *t-Student*, pode-se calcular o intervalo de predição. Este intervalo fornece a gama ao redor da estimativa, dentro da qual é provável que o tamanho do programa atual caia. A fórmula para o intervalo de predição é:

$$\text{Gama} = t(a/2, n\text{-}2)s\sqrt{\left(1 + 1/n + \left((x_k - x_{avg})^2 / \sum_{i=1}^{n}(x_i - x_{avg})^2\right)\right)}$$

Figura 4-10 - Intervalo de Predição

Aqui, os termos x_i são novamente os números de LOC objeto estimadas em cada programa no banco de dados histórico. O termo x_{avg} é a média de LOC objeto estimadas nestes mesmos programas. O termo x_k são as LOCs objeto estimadas no programa novo, e *n* é o número de programas no banco de dados. $a/2$ refere-se à porcentagem usada para o intervalo de predição, como 70 ou 90 por cento. Note-se que tabelas estatísticas padrões dão a distribuição de *t* de um lado só, embora a fórmula gama use os valores dos dois lados da distribuição *t*. A distribuição de dois lados é indicada pelo parâmetro $a/2$ na expressão *t*. Em geral, para achar a distribuição *t* para um $a/2$ de 70 por cento, ver a Tabela A-96 debaixo de p = 0.85 e para 90 por cento, debaixo de p = 0.95.

O termo s na equação é o desvio padrão dos dados ao redor da linha de regressão. É calculado como mostrado na equação abaixo, usando os parâmetros ß0 e ß1.

$$\text{Variância} = s^2 = (1/(n-2)) \sum_{i=1}^{n} (y_i - ß0 - ß1)^2$$

Figura 4-11 - Variância

:

$$\text{Desvio padrão} = \sqrt{(\text{Variância})} = s$$

Agora é possível calcular o valor da Gama. O intervalo de predição é então o valor estimado calculado para LOCs do programa mais ou menos esta gama de intervalo.

Categorias de Objetos

O método de estimativa PROBE requer que se tenha dados históricos dos tamanhos dos objetos desenvolvidos e que estes dados sejam divididos em categorias. A seguir são descritos **métodos para dividir esses dados**.

São necessárias categorias de tamanho de objeto para se ter uma estrutura para julgar o tamanho dos objetos novos no produto planejado.

Como se está principalmente interessado no tamanho relativo dos objetos, com base no julgamento da sua complexidade funcional, é útil normalizar o tamanho dos objetos dividindo o total de LOC objeto pelo número de métodos em cada objeto. Agora um objeto complexo com um método pode ser distinguido de um objeto simples com muitos métodos.

Gamas de tamanho de objeto

Para julgar o tamanho relativo de objetos, usa-se o desvio padrão:

$$\text{Variância} = s^2 = (1/n) \sum_{i=1}^{n} (x_i - y_{avg})^2$$

$$\text{Desvio padrão} = \sqrt{(\text{Variância})} = s$$

Onde n é igual ao número de itens, x_i é igual ao número de LOCs por método e x_{avg} é igual à média.

Exemplo:

Tabela 4-3 - LOC por Método e Desvio Padrão de Objetos Texto em Pascal

Nome do Objeto	Numero de Metodos	LOC Objeto	LOC por Método	$(LOC-LOC_{avg})^2$
each_line	3	31	10,333	93,494
each_char	3	18	6,000	196,072
list_clump	4	87	21,750	3,054
character	3	87	29,000	80,954
single_character	3	25	8,333	136,171
string_read	3	18	6,000	196,072
list_clp	4	89	22,250	5,051
char	3	85	28,333	69,402
single_char	3	37	12,333	58,817
converter	10	558	55,800	1.281,456
string_manager	4	82	20,500	0,247
string_builder	5	82	16,400	12,978
	10	230	23,000	8,985
Total			260,033	2.142,753
Média			20,003	
Variância = Total/n				164,827
Desvio Padrão = $(Variância)^2$				12,839

No caso acima, o desvio padrão é de 12,839, o que significa que os pontos centrais da gama de tamanho são os seguintes:

Muito Pequeno (VS) = -5.68 (=20 – 2*12,839)
Pequeno (S) = 7.16 (=20-12,839)
Médio (M) = 20.0
Grande (L) = 32.84 (=20+12,839)
Muito Grande (VL) = 45.68 (=20+2*12,839)

Obs: 20 = média.

O método anterior pode resultar em número negativo de LOCs, então usa-se o método seguinte:

1. Calcular o logaritmo natural do valor de LOC por método para cada um dos 13 objetos. Por exemplo, o logaritmo natural do primeiro objeto, ***each_line***, 10,333, é 2,335.
2. Calcular a média dos valores logarítmicos. Neste caso, a média é 2.802.

3. Calcular a variância dos valores logarítmicos ao redor do valor do meio ou média. Faz-se isto como segue:
   - Para cada termo, calcular o quadrado de sua distância do meio; por exemplo, para o primeiro termo:
   $$[\ln 10{,}333 - (\ln x)_{avg}]^2 = (2{,}335 - 2{,}803)^2 = (0{,}467)^2 = 0{,}2173$$
   Notar que 2,802 não é o log da média, é a média dos logs.
   - Somar todos os valores quadrados para dar 5,2348.
   - Dividir pelo número de objetos, 13, para dar a variância de 0,4027.
   - Tomar a raiz quadrada da variância, 0,4027, para obter o desvio padrão, 0.6346.

4. Usando estes valores logarítmicos para média e desvio padrão, calcular os logaritmos dos pontos centrais da gama de tamanho. Exemplo:
   $\ln (VL) = avg_{\ln} + 2*$ DesvioPadrao = 2,802 + 1,269 = 4,071
   $\ln (L) = avg_{\ln} + 1*$ DesvioPadrao = 2,802 + 0,635 = 3,437
   ln (M) = 2,802
   ...

5. Tomar os antilogaritmos destes para obter a os pontos centrais da gama de tamanho de objeto em LOCs. Exemplo:

   $e^{4,071}$ = 58,62 LOC
   $e^{3,437}$ = 31,09 LOC
   $e^{2,802}$ = 16,48 LOC
   ..

Esse método funciona porque os valores dos logs dos tamanhos dos objetos são mais de perto distribuídos normalmente do que os valores de tamanhos de LOCs.

Inicialmente, com poucos dados, deve-se fazer esses cálculos para todos os objetos. Quando houver dados suficientes para uma distribuição por categorias, deve-se dividir os dados em grupos e fazer os cálculos para cada grupo.

4.4.2.1 Distribuição de tamanho de objeto

Deve-se balancear as estimativas de forma a que os tamanhos das categorias se conformem mais ou menos à distribuição normal, conforme a figura a seguir.

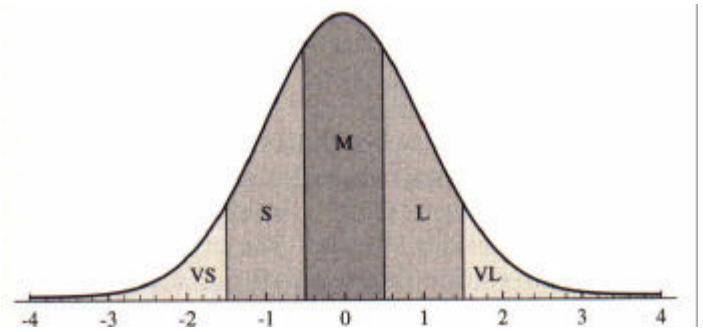

Figura 4-12 - Distribuição normal com gama de tamanho

Os valores da gama são os seguintes:

- 6.68 por cento devem ser muito pequenos,
- 24.17 por cento devem ser pequenos,
- 38.3 por cento devem ser médios,
- 24.17 por cento devem ser grandes,
- 6.68 por cento devem ser muito grandes.

Se a estimativa tiver uma tal distribuição, seria típico dos programas do banco de dados de estimativa de tamanho do desenvolvedor.

Usar estes números apenas como guia geral. Se for muito diferente das estimativas atuais, reexaminar para ver se há algum erro.

4.4.2.2 Considerações sobre estimativas

Apesar de, geralmente, a taxa de erros do estudante ao fazer estimativas parecer não melhorar com os exercícios do PSP, é possível melhorar a habilidade de estimativa geral mesmo que cada estimativa esteja substancialmente errada.

À medida que se aprende a usar dados de tamanho históricos, a qualidade deve melhorar. Ao quantificar a experiência de estimar, é possível fazer melhores estimativas e compensar as tendências pessoais. Por exemplo, se o engenheiro sempre estima para menos, ele logo irá reconhecer esta tendência e começar a corrigi-la. Com o *feedback* de cada estimativa, é possível ajustar o processo de estimativa para reduzir gradualmente as tendências.

Quando os parâmetros ß0 e ß1 convergirem para valores estáveis, pode-se parar de recalculá-los a cada estimativa. À medida que o processo evoluir, contudo, é preciso atualizar os cálculos estatísticos para representar as práticas correntes.

Quando os parâmetros de regressão linear parecerem inaceitáveis, deve-se usar os valores médios nas estimativas. Por exemplo:

LOC (de 4 programas) estimadas: 427
LOC (atual desses 4 programas): 583
Taxa = 583/427 = 1,365
LOC estimadas (do programa atual): 137
LOC estimadas corrigido pela taxa: 187.

Antes de 3 conjuntos de dados históricos não de seve usar o PROBE. Nesses casos, **calcular os parâmetros ß0 e ß1 com o método da média**.

Só é possível estimar corretamente o tamanho do produto após completar o projeto.

Tendências

Em **grandes projetos**, deve-se adquirir o hábito de fazer novas estimativas de tamanho e recursos ao completar o projeto e ao completar o código, e comparar essas valores na fase de *Postmortem*.

Selecionando um nível de abstração

Quanto menor for a abstração, maior será o número de pontos de dados para cada projeto. É preciso cuidado ao selecionar o nível de abstração, pois se um objeto típico tem métodos com uma média de 10 a 15 LOCs e existem de 3 a 10 métodos por objeto, então os objetos variarão de 30 a 100 LOCs cada. Quando o PROBE for usado em um sistema de 50.000 LOC, será necessário categorizar, identificar e taxar a complexidade de cerca de 400 a 600 objetos. Isto pode dar muito trabalho.

Com o PROBE, um programa de 5.000 LOCs pode ser estimado em 1 ou 2 horas. Então, 50.000 LOCs podem ser estimadas em 100 horas, ou 13 dias, mais ou menos. Entretanto, um programa desse tamanho deve levar muitos meses de programação.

Grandes Sistemas: nestes casos, pode ser desejável usar construções de alto nível como *proxies*, agrupando grupos de objetos em classes. Entretanto, isso pode comprometer as vantagens estatísticas de usar rotinas suportadas por estimativas de dados, devido às individualidades (diferenças) dos objetos. Uma abordagem prática para resolver isso é dividir o programa em diversas partes principais, estimando cada parte e usando o nível de abstração de objetos.

4.4.3 PSP1 - Conteúdos do processo

O PSP1 introduz a estimativa de tamanho ao PSP. Antes de começar um desenvolvimento de *software* do PSP1, o método de estimativa de tamanho PROBE deve ser usado para estimar o tamanho das LOCs novas e alteradas no novo programa. Também, devem ser estimadas as LOCs base, reusadas, apagadas, modificadas, adicionadas e total de novas e mudadas. Notar que quando o PSP é usado primeiramente, não existirão dados históricos para fazer estimativas de tamanho. Assim, deve-se estimar as LOC objeto e deve-se usar o próprio julgamento para obter o total de LOCs novas e mudadas. Para o segundo e o terceiros programas escritos com o PSP1, é possível usar os dados do primeiro e do segundo para guiar as estimativas. O *Script* de Estimativa PROBE da Tabela A-45 guia esses passos.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Em adição aos objetivos do PSP0.1, o PSP1 é planejado para estabelecer um procedimento ordenado e repetível para estimativas de tamanho de desenvolvimento de *software*. À medida que este processo é usado para calcular o desenvolvimento de programas, será construída uma base crescente de dados estimados que devem ajudar a fazer estimativas de tamanho progressivamente mais precisas.

As condições prévias para o PSP1 são o PSP0.l e o programa 3A. É necessário, também, ter os dados de tamanho atuais de pelo menos três programas previamente desenvolvidos.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP1 - *Script* de Processo	**Tabela A-39**
PSP1 - *Script* de Planejamento	**Tabela A-40**
PSP1 - *Script* de Desenvolvimento	**Tabela A-41**
PSP1 - *Script* de *Postmortem*	**Tabela A-42**
PSP1 - Resumo de Plano de Projeto e Instruções	**Tabelas A-43 e A-44**
***Script* de Estimativa PROBE**	**Tabelas A-45**
Modelo de Relatório de Teste e Instruções	**Tabelas A-46 e A-47**
Modelo de Estimativa de Tamanho e Instruções	**Tabelas A-48 e A-49**
Proposta de Melhoria de Processo (PIP) e Instruções	Tabelas A-37 e A-38
Padrão de codificação	Tabelas A-13
Log Registro Tempo e Instruções	Tabelas A-7 e A-8
Log Registro Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ELEMENTOS DE PROCESSO NOVOS

Modelo de Relatório de Teste. O Modelo de Relatório de Teste é mostrado na Tabela A-46 e suas instruções na Tabela A-47. É usado para registrar dados dos testes. Ao incorporar programas pequenos em programas maiores, fazer modificações, corrigir

problemas, ou ao reusar os programas, será útil reexibir os testes previamente completados. Para fazer isto, é necessário saber quais testes foram feitos, quais dados de teste foram usados e os resultados que foram obtidos.

Isso pode ser útil ao se modificar um programa para incorporar uma nova função. Como parte dos testes, o desenvolvedor poderá se assegurar de que as mudanças não causaram problemas com funções que já funcionavam previamente. Tais problemas são chamados de regressões. Problemas de regressão são comuns quando são feitas muitas mudanças em grandes programas com multi-versões. Um teste de regressão é mais facilmente e efetivamente feito rodando de novo os testes que funcionavam previamente. Porém, isto é difícil se não foram salvos os dados de teste adequados.

ESPECIFICAÇÃO DE RELATÓRIO DE PROCESSO

Para os exercícios feitos com o PSP1 e processos de níveis mais altos, é necessário satisfazer os critérios de avaliação usados para o PSP0.1:

- Os dados de processo estão completos.
- Os dados são precisos e autoconsistentes.
- O relatório de processo é submetido na própria ordem e formato.

Se estes critérios não são satisfeitos, os dados não serão satisfatórios para as análises requeridas nos exercícios posteriores. Com o PSP1, um quarto critério de avaliação é que o engenheiro efetivamente use seus dados históricos para guiar seu trabalho. Por exemplo, a distribuição de tempo planejada deve se relacionar à experiência histórica.

Os itens a serem incluídos nos relatórios de exercício do PSP1 e a ordem nas quais eles serão submetidos são:

- **Sumário do Plano de Projeto do PSP1**
- **Modelo de Relatório de Teste**
- Os formulários PIP, inclusive uma declaração breve das lições aprendidas
- **O Modelo de Estimativa de Tamanho**
- Log de Registro de Tempo
- Log de Registro de Defeito
- Listagem do Programa Fonte

4.4.4 Exercícios

Existem dois exercícios para esta seção. A primeira tarefa usa o **PSP0.1 para escrever o programa 3A**. Também requer produzir o **Relatório de Análise de Defeito, R3**. A segunda tarefa usa o **PSP1 para escrever o programa 4A**.

Programa 3A

Programa 3A - Condições prévias e Referências: Seção 4.4 e o programa 2A.

Programa 3A - requerimentos: escrever um programa para contar as LOC totais de programas, LOCs totais de cada objeto que o programa contém e o número de métodos em cada objeto. Produzir um contador de LOCs para um arquivo fonte inteiro e um contador separado de métodos e LOCs para cada objeto. Imprimir cada nome de objeto junto com suas LOCs e o método de contagem. Também imprimir a contagem total de LOC do programa. **Se uma linguagem objeto-orientada não é usada, contar as LOCs de procedimentos e funções e imprimir os nomes dos procedimentos e funções e a contagem de LOC.** Usar o padrão de contagem produzido para o exercício de relatório R1. É aceitável melhorar o programa 2A ou reusar alguns de seus métodos, procedimentos, ou funções no desenvolvimento do programa 3A.

Programa 3A - teste: testar completamente o programa. No mínimo, testar o programa contando as LOC totais do programa e objetos nos programas 1A, 2A e 3A. Incluir no relatório de teste uma tabela que dá as contas obtidas com o programa 2A e 3A para todos os programas escritos até a data. Usar o formato no exemplo na Tabela A-15(a). Se não se está usando programação orientada a objetos, usar o formato de relatório na Tabela A-15(b).

R3 - O Relatório de Análise de Defeito

Objetivos da tarefa:

- Entender a densidade e os tipos de defeitos introduzidos e achados ao se desenvolver os programas iniciais deste trabalho;
- Demonstrar a importância de juntar cuidadosamente, registrar e relatar dados de processo.

Tarefa: analisar os defeitos encontrados no desenvolvimento dos programas iniciais e produzir um relatório que inclua o seguinte:

1. Um tabela informando:

   - o número total de defeitos achados,
   - as LOCs novas e alteradas, e
   - os defeitos por KLOC.

Fornecer estes dados por cada programa e para o total dos programas escritos *Até a Data.* Usar o formato na Tabela A-26.

2. Um tabela informando:

   - o número de defeitos achados na compilação,

- o número de defeitos achados nos testes,
- o número de defeitos por KLOC achados na compilação, e
- o número de defeitos por KLOC achados nos testes.

Fornecer estes dados por cada programa e para o total dos programas escrito *Até a Data.* Usar o formato na Tabela A-27.

3. Produzir uma tabela que mostre a média de tempo de correção para os defeitos achados na compilação, defeitos achados nos testes, defeitos injetados no projeto, e defeitos injetados na codificação. Produzir uma tabela no formato mostrado na Tabela A-28.

Resultado: completar e submeter as tabelas exigidas.

Cronograma: fazer esta tarefa junto com ou imediatamente após a conclusão do programa 3A. Esta tarefa não deve ser feita após aquele ponto porque um dos objetivos da tarefa é demonstrar a importância de se juntar dados de qualidade de processo.

Programa 4A

Programa 4A - Condições prévias e Referências: programa 1A.

Programa 4A - requerimentos: escrever um programa para calcular os parâmetros de regressão linear de estimativa de tamanho para um conjunto de *n* programas onde dados de LOC objeto históricas e LOC novas e alteradas estão disponíveis. Aumentar a lista encadeada do programa 1A para guardar os registros de dados de *n*, onde cada registro guarda dois números reais.

Programa 4A - teste: testar completamente o programa. No mínimo, usar este programa para calcular os parâmetros ß para três casos. Para o primeiro, usar os dados na Tabela A-16 para LOC objeto estimadas e LOCs novas e mudadas atuais. Os valores resultantes devem ser ß0 = -22.55 e ß1 = 1.7279. Segundo, calcular os parâmetros ß0 e ß1 para a regressão ajustada das colunas LOCs novas e alteradas estimadas e LOCs novas e alteradas atuais na Tabela A-16. A resposta neste caso deve ser ß0 = -23.92 e ß1 = 1.4310. Terceiro, calcular os parâmetros ß0 e ß1 para as LOCs novas e mudadas estimadas e as LOCs atuais novas e mudadas para os programas 2A, 3A, e 4A desenvolvidos. Preparar um relatório dos testes que incluam uma tabela, destes testes, dos resultados planejados e atuais no formato na Tabela A-17.

4.5 PSP1.1 – Planejamento de tarefas e tempo

4.5.1 Estimando recursos e horários

É possível produzir planos melhores, por menos tempo, estruturando o processo de planejamento, juntando dados históricos e usando métodos efetivos. À medida que o desenvolvedor ganha experiência de planejamento, sua confiança em suas estimativas lhe dará a convicção para defender e vender seus planos. Esta melhora da habilidade de planejamento também melhorará as relações de trabalho com os colegas e gerentes e o tornará um engenheiro de *software* mais consistente.

O processo de planejamento inicia com uma estimativa de tamanho. Então, os recursos[13] exigidos são estimados e um cronograma é produzido. Para fazer isso, relacionam-se as horas de desenvolvimento usadas em projetos anteriores às estimativas de tamanho atuais. Como todo projeto é diferente, se todas as atividades de projeto forem amontoadas em algum número de produtividade global, muito do pensamento exigido para produzir um bom plano será eliminado. Assim, é importante refletir sobre o que é requerido exclusivamente para cada projeto. A figura abaixo resume o procedimento de estimativa de tempo de desenvolvimento.

Estimando o tempo de desenvolvimento:

Basicamente depende da quantidade e qualidade dos dados históricos:

Produtividade:
- opção 1: usar o cálculo de regressão sobre os dados de LOC objeto estimadas e horas atuais (método PROBE).
- opção 2: usar LOCs atuais novas e alteradas e horas de desenvolvimento atuais para fazer o cálculo de regressão.
- opção 3: existem dados históricos, não existem dados de LOC objetos estimadas e não existem dados para regressão: calcular a produtividade em LOCs por hora.
- opção 4: julgamento.

Tempo:

Após calcular a produtividade, calcular o tempo requerido para o programa, dividindo as LOCs alteradas e novas estimadas pela taxa de produtividade.

Intervalo:

Na opção 3, calcular o maior e o menor tempo usando os dados históricos, ou seja, calcular a produtividade do maior tempo e do menor tempo, e aplicar essas duas taxas de produtividade ao tamanho estimado do programa atual. Nas opções 1 e 2 fazer os cálculos do intervalo pelo método do Intervalo de Predição (UPI e LPI)

Figura 4-13 - Estimando o tempo de desenvolvimento.

[13] No PSP o recurso planejado é o tempo: quanto tempo se gasta desenvolvendo um programa.

Antes de se poder fazer um cronograma, é necessário um plano de recursos detalhado. Geralmente, esta seria uma estimativa de quantas horas diretas se espera gastar em cada tarefa. É preciso, também, projetar o tempo direto total que estará disponível. Esta projeção tem que permitir os outros compromissos e as muitas atividades subordinadas que preenchem um dia de trabalho normal. Os engenheiros freqüentemente acham que eles só gastam aproximadamente 50 por cento do seu tempo em seus trabalhos diretos de projeto.

Com uma estimativa do tempo direto disponível e um plano de recursos detalhado, é possível calcular quando cada tarefa começará e quando terminará. Então são fixadas as datas de planejamento para os marcos de projeto fundamentais. Estas datas provêem uma base sã para se fazer compromissos.

O monitoramento de valor merecido (ou ganho) é um modo para se avaliar o progresso do projeto. Com ele é estabelecido um valor relativo para cada tarefa e creditado esse valor (valor merecido) quando a tarefa é completada.

O sistema de valor merecido[14] provê uma escala de valor comum para cada tarefa, independentemente do tipo de trabalho envolvido. São estimadas as horas totais para fazer o trabalho inteiro e fornecido para cada tarefa um valor de planejamento, baseado em sua percentagem estimada deste total. O crédito de valor merecido é dado, então, a essa tarefa, quando completada. Tarefas parcialmente completadas não adquirem nenhum crédito.

4.5.2 PSP1.1 - Conteúdos do processo

O processo PSP1.1 introduz a estimativa e planejamento de recursos e horários. Quando combinado com o método de estimativa de tamanho PROBE - introduzido com o PSP1 - e com suficientes dados de tamanho e custo, será possível, então, fazer melhores planos de desenvolvimento e julgar a precisão desses planos.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Em adição aos objetivos do PSP1, os objetivos do PSP1.l são introduzir e praticar métodos para:

- fazer planos de recursos e horário,
- monitorar o desempenho pessoal contra estes planos, e
- julgar datas prováveis de conclusão de projeto.

A condição prévia para o processo PSP1 .1 é o PSP1 e a presente seção.

[14] O sistema de valor merecido é simplesmente uma porcentagem do tempo total estimado para o programa, atribuída à tarefa.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP1.1 - *Script* de Processo	**Tabela A-50**
PSP1.1 - *Script* de Planejamento	**Tabela A-51**
PSP1.1 - *Script* de desenvolvimento	**Tabela A-52**
PSP1.1 - *Script* de *Postmortem*	**Tabela A-53**
PSP1.1 - Sumário do Plano de Projeto e Instruções	**Tabelas A-54 e A-55**
Modelo de Planejamento de Tarefa e Instruções	**Tabelas A-56 e A-57**
Modelo de Planejamento de Horário e Instruções	**Tabelas A-58 e A-59**
***Script* de Estimativa PROBE**	**Tabelas A-45**
Modelo de Relatório de Teste e Instruções	**Tabelas A-46 e A-47**
Modelo de Estimativa de Tamanho e Instruções	**Tabelas A-48 e A-49**
Proposta de Melhoria de Processo (PIP) e Instruções	**Tabelas A-37 e A-38**
Padrão de Codificação	Tabelas A-13
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

NOVOS ELEMENTOS DE PROCESSO

Os dois novos elementos de processo incluídos com o PSP1.1 são o Modelo de Planejamento de Tarefa e o Modelo de Planejamento de Horário. Estes são tipicamente usados para projetos que levarão vários dias ou semanas para terminar.

A seção sumária do Resumo de Plano de Projeto foi ampliada para incluir estatísticas de processo básicas para este programa e para o trabalho de desenvolvimento até a data. Conseqüentemente, o desenvolvedor pode comparar seu recente desempenho com sua média histórica e julgar a racionalidade dos seus planos.

ESPECIFICAÇÕES DE RELATÓRIO DE PROCESSO

O processo PSP 1.1 usa os mesmos critérios de avaliação usados para PSP 1:

- Os dados de processo estão completos,
- Os dados são precisos e autoconsistentes,
- O relatório de processo é submetido na própria ordem e formato,
- Os dados históricos são usados no planejamento do trabalho.

Os itens a serem incluídos nos relatórios de exercício PSP1.1 e a ordem nas quais eles serão submetidos são:

- Sumário do Plano de Projeto do PSP1.1,
- Modelo de Relatório de teste,
- O formulário PIP, inclusive com uma breve declaração das lições aprendidas,
- O Modelo de Estimativa de Tamanho,
- O Modelo de Planejamento de Tarefa,
- O Modelo de Planejamento de Horário,

- Log de Registro de Tempo,
- Log de Registro de Defeito,
- Listagem do Programa Fonte,

4.5.3 Exercícios

O plano de curso padrão para esta seção usa o PSP 1.1 para escrever o programa 5A.

Programa 5A

Requerimentos do Programa 5A: escrever um programa para integrar numericamente uma função que usa a regra de Simpson e escrever a função para a distribuição normal. O programa deve ser projetado para integrar usando várias funções providas. Este programa será necessário para calcular os valores das várias distribuições estatísticas usadas nas tarefas de programa posteriores e na análise dos dados do PSP. Este programa é usado em vários exercícios subseqüentes onde são usados programas para calcular valores da distribuição normal, a distribuição *t* e a distribuição χ^2.

Programa 5A - teste: testar completamente o programa. Incluir um teste para calcular mais tarde os valores de probabilidade da distribuição integral normal de $-\infty$ até x = 2.5, de $-\infty$ até x = 0.2 e de $-\infty$ até x = - 1.1. Os resultados devem ser, aproximadamente, 0.9938, 0.5793, e 0.1357, respectivamente. Incluir no relatório de teste uma tabela de resultados no formato da Tabela A-18.

4.5.4 Medidas no PSP

Para ajudar a entender e melhorar os processos, dados são coletados e analisados. Para poder fazer perguntas inteligentes sobre o seu processo, o engenheiro precisa de um *modelo* de processo. Uma vez que a definição do processo de *software* provê o modelo de obtenção de dados, ao menos uma definição de processo básica deve sempre preceder a coleta dos dados. Esta é a razão pela qual o processo de *software* é definido antes de se juntar dados extensos.

As medidas usadas para colher dados caem em uma de três categorias: produto, processo e recurso. Medidas de produto referem-se ao volume do produto produzido. Medidas de processo relacionam eventos às fases de processo, isto é, quantificam o comportamento do processo (contagem de eventos, medição de tempo, etc). Recurso mede principalmente o tempo do programador (horas de trabalho). Embora a medida comum de recursos seja de meses ou semanas, Humphrey reafirma aqui que o tempo pessoal deve ser medido em minutos (para o controle das distrações, como telefonemas, cafezinhos, entre outros).

Objetivos gerais da obtenção de dados no PSP:

- entender como o processo funciona;
- determinar os passos a serem tomados para melhorar a qualidade do produto;
- determinar o impacto da mudança de processo na produtividade pessoal;
- estabelecer *benchmarks* para medir a melhoria do processo;
- ajudar a fazer planos mais acurados.

Medidas apropriadas são importantes para melhorar a estimativa, prover objetivos e monitorar a informação. Para alcançar validade, as medidas do PSP foram definidas usando a abordagem GQM, que tenta definir medidas válidas identificando as **metas** da medida; as **questões** que investigam realização dessas metas; e **medidas** que possam responder a essas perguntas.

O paradigma Métrica-Questão-Objetivo (*Goal-Question-Metric*-GQM) pode ajudar a projetar e implementar um programa de medida. Dados são coletados para conhecer as metas específicas e responder a questões explícitas. Estes dados também devem ser precisamente definidos, consistentemente obtidos e corretamente usados.

Exemplo de GQM:

- **Objetivo: produzir programas sem defeitos.**
- **Questão: como produzir *software* de tal qualidade que não sejam encontrados defeitos em testes ou usos posteriores?**

As ajudas requeridas para, manualmente, colher, analisar e usar dados do PSP são os formulários, bancos de dados, planilhas eletrônicas e relatórios sumários. Ferramentas automatizadas podem ajudar a fazer estas coletas de dados e análises mais convenientes, oportunas e precisas. Porém, como o julgamento é envolvido ao colher a maioria dos dados de processo, ferramentas para colher automaticamente estes dados não são prováveis no futuro próximo[15]. Por exemplo, quando o desenvolvedor pára de digitar, o que está acontecendo? Ao mudar o código, ele está corrigindo um defeito ou melhorando o programa? Onde o defeito foi injetado?

A obtenção de dados sobre o processo de *software* causa impacto no desempenho. Também pode ser uma tarefa consumidora de tempo e tediosa. Assim, se não se está comprometido a juntar e usar os dados, os dados não serão coletados ou serão incompletos e inexatos.

Ao começar a usar o processo pessoal, provavelmente haverá curiosidade para saber se o desempenho está melhorando. Embora esta seja uma preocupação válida, é quase impossível de resolver sem dados significativos. Somente é possível conhecer as melhorias ao se examinar estatisticamente um volume grande de dados.

[15] Aqui, mais uma vez, Humphrey manifesta-se contrariamente à automatização do PSP.

Os dados de desempenho pessoal podem ser desencorajadores. Os resultados variarão, e, à medida que se esforce para melhorar, eles podem não demonstrar nenhuma melhora consistente. Para alcançar uma tendência de melhora consistente, é necessário fazer mudanças específicas no processo pessoal.

Um *processo básico* fornece uma base para analisar futuros resultados e determinar onde e como se está fazendo progresso. Fazer isto ajuda a planejar o trabalho e justifica os esforços de melhoria de processo.

4.5.5 Exercícios

O plano de curso padrão para esta seção usa o PSP1.1 para escrever o programa 6A. Este é o mesmo processo usado para o programa 5A.

Programa 6A

Programa 6A - condições prévias: programas 4A e 5A.

Requerimentos do Programa 6A: escrever um programa para calcular uma estimativa de LOCs e os intervalos de predição de 90 e 70 por cento para esta estimativa. Usar o programa 5A para calcular o valor da distribuição *t* e usar uma lista encadeada para os dados. O programa 4A pode ser aumentado para se desenvolver este programa. Notar que para calcular o valor *t*, integra-se de 0 até um valor de teste de *t*. Acha-se o valor correto ajustando sucessivamente o valor de teste de *t* para cima ou para baixo até que o valor de *p* esteja dentro de um erro aceitável de 0.85 (para um intervalo de predição de 70 por cento) ou 0.95 (para um intervalo de predição de 90 por cento).

Programa 6A - teste: testar completamente o programa. Como um teste, usar os dados para LOC objeto estimadas e LOC atuais novas e mudadas na Tabela A-16 e os valores ß0 e ß1 achados no teste do programa 4A. Também assumir um valor de LOC objeto estimadas de 386. Sobre essas condições, os valores do LOC estimadas e os valores de parâmetro obtidos devem ser como segue:

ß0 = -22.55
ß1 = 1.7279
S = 197.8956
Projeção = 644.429
t(70 por cento) = 1,108
t(90 por cento) = 1,860
Gama (70 Por cento) = 229,9715, UPI = 874,401, LPI = 414,4579
Gama (90 Por cento) = 386,0533, UPI = 1030,483, LPI = 258,3761

Usando as LOC objeto estimadas desenvolvidas ao planejar o programa 6A, calcular as LOC projetadas novas e alteradas e o intervalo de predição para o programa

6A. Comparar este intervalo com as LOC atuais novas e alteradas resultantes. Incluir estes dados no relatório de teste que usa o formato da Tabela A-19.

4.6 PSP2 - Melhorando a qualidade

4.6.1 Revisão de projeto e código

O propósito das revisões é assegurar que os programas produzidos sejam da mais alta qualidade. Dos muitos tipos de revisões que podem ser executadas, as principais são as inspeções, *walk-throughs* e revisões pessoais. Podem ser usadas revisões nos requerimentos, projeto, documentação ou qualquer outro elemento de produto. Esta seção trata de revisão de projeto e código.

Muitos projetos de *software* gastam quase a metade do seu tempo de desenvolvimento em testes. Isto é muito ineficiente. A revisão de projeto e código é um modo muito mais eficiente para achar e corrigir defeitos. Com as revisões, é possível achar os defeitos diretamente, enquanto que em testes obtém-se somente os sintomas. A diferença crucial entre estas duas abordagens é chamada depuração. Ao revisar um programa, o engenheiro sabe onde está e o que, supostamente, a lógica está fazendo. Assim, as correções serão, provavelmente, mais completas e corretas.

As revisões do PSP irão obter um retorno maior que qualquer outra coisa que se possa fazer. Cada programa deve ser lido, estudado e entendido. Devem ser corrigidos os defeitos de lógica, estrutura e clareza. Quando algumas áreas estiverem obscuras ou confusas, deve-se adicionar comentários ou, melhor ainda, fazer uma reescrita completa. O engenheiro deve fazer os programas fáceis de se ler e entender. Deve produzir algo que ele ficaria orgulhoso de publicar ou mostrar a seus amigos e colegas.

Os três princípios básicos das revisões pessoais são: estabelecer metas de revisão, seguir um processo disciplinado e medir e melhorar este processo. A decisão para fazer revisões é dirigida pelo desejo de ser produtivo e produzir produtos de qualidade. Para saber se as revisões o estão ajudando a alcançar estas metas, o desenvolvedor precisa medi-las.

Aqui é onde se usa uma distribuição de Pareto para estabelecer prioridades de revisão e desenvolver uma lista de conferência de revisão. Seria sábio reexaminar esta distribuição de Pareto periodicamente, para assegurar-se que ainda se está focalizando corretamente os defeitos mais significativos do processo.

Princípios de revisão de Projeto:

- produzir projetos que possam ser revisados;

- seguir uma estratégia explícita de revisão;
- revisar o projeto em estágios;
- verificar se a lógica implementa corretamente os requerimentos.

Embora o rendimento dê a melhor medida da qualidade da revisão, não se pode calculá-lo antes do desenvolvimento estar completo. Assim, é essencial achar medidas atuais ou imediatas que se relacionem ao rendimento, como LOCs revisadas por hora.

Exercícios

O plano de curso padrão para esta seção requer a produção do relatório R4, o relatório central, neste momento. Também devem ser produzidas uma lista de conferência de revisão de projeto e uma lista de conferência de revisão de código para uso com o PSP2.

R4 - O RELATÓRIO DE ANÁLISE CENTRAL

Avaliação de tarefa: definir um processo para analisar os dados do PSP e produzir um relatório. Usar este processo para produzir o relatório R4. Este processo será atualizado para produzir o relatório final ao término do curso.

Objetivos de tarefa:

- Adquirir experiência na definição e uso de um processo,
- Entender a base do processo de *software* pessoal e como ele tem mudado no curso,
- Prover listas de conferência de revisão de código e projeto.

Tarefas:

Tarefa 1 - Desenvolver um processo para analisar os dados e criar um relatório sobre os programas 1A até 6A. Isto deve incluir os resultados listados a seguir, debaixo da Tarefa 3. Este processo tem que incluir uma fase de planejamento, as fases de desempenho de tarefa, e uma fase de *Postmortem*. Produzir e submeter um *script* de processo e formulários de planejamento para ordenar este processo.

Tarefa 2 - Planejar e ordenar o processo definido na Tarefa 1. Usar o formulário de planejamento para registrar o tempo planejado para este trabalho e monitorar e registrar o tempo atual gasto. Submeter os dados de processo planejados e atuais junto com o relatório de análise.

Tarefa 3 - Analisar os dados dos programas 1 até 6. Análises de planilha eletrônica são sugeridas e resumos gráficos e apresentações são encorajadas. No mínimo, produzir o que segue:

- Uma análise de LOC e a precisão da estimativa do tempo de desenvolvimento e como ela evoluiu durante os programas desenvolvidos até a data;
- Uma análise dos tipos de defeito injetados e removidos nos programas desenvolvidos até a data. Estes dados devem ser mostrados em um formato semelhante a Tabela A-29;
- Uma análise dos tipos de defeitos achados pelo compilador. Estes dados devem ser mostrados em um formato semelhante à Tabela A-30;
- Uma análise dos tempos de correção de defeito, usando o formato usado no relatório R3 (Tabela A-28);
- Desenvolver uma lista de conferência de revisão de projeto para achar os defeitos de projeto mais freqüentes em uma revisão de projeto;
- Desenvolver uma lista de conferência de revisão de código para achar os mais freqüentes defeitos de código em uma revisão de código.

Resultado: usar o método GQM para esboçar conclusões e estabelecer metas de melhoria pessoal. Submeter as análises exigidas, tabelas e listas de conferência junto com um relatório sumário escrito. Usar gráficos onde quer que seja possível.

Cronograma: fazer esta tarefa junto com o programa 6A ou logo após sua conclusão. Preceder o desenvolvimento de qualquer programa usando o processo PSP2 ou posteriores.

4.6.2 Gerenciamento da qualidade de *software*

À medida que o engenheiro de *software* trabalha para melhorar a qualidade do *software* desenvolvido, ele deve se focalizar na habilidade do seu processo para produzir produtos de qualidade. Deve buscar os métodos mais efetivos para localizar os defeitos, como também os modos mais efetivos para os prevenir. Também reconhecer que, quanto mais tempo são deixados no produto, os custos de achar e corrigir defeitos escalam rapidamente. A melhor estratégia é assegurar que o programa seja da mais alta qualidade ao produzi-lo inicialmente.

Bugs de *software*, defeitos e erros são só uma pequena faceta da qualidade do *software*; porém, este é o foco de qualidade do PSP. Os defeitos raramente são a prioridade de qualidade principal dos usuários, mas eles são um foco essencial do PSP porque os defeitos são mais bem controlados ao nível individual. Se os programas elementares em um sistema tiverem muitos defeitos, o desenvolvimento do sistema inteiro será prejudicado pelo processo demorado e caro de achar e corrigir esses defeitos.

Essencialmente, a qualidade de *software* é um assunto de economia. Porém, poucas organizações têm os dados sobre os quais fundar bons planos de qualidade. A medida básica de qualidade do PSP é o rendimento - a porcentagem de defeitos removidos antes do primeiro teste ou compilação.

Os componentes principais do custo de qualidade são os custos de falhas, custos de avaliação e custos de prevenção. A medida do custo de qualidade é tipicamente usada para avaliar o desempenho da qualidade das organizações. Quando usada em projetos individuais, mostrará uma variação considerável. Uma medida de custo de qualidade também pode ser usada com o PSP.

Técnicas de benchmarking são úteis para comparar processos. Elas podem ser usadas para monitorar processos através do tempo ou para comparar processos. Para fazer isto, é necessário definir medidas de processo que são independentes do processo, mas que reflitam sua capacidade e robustez. Infelizmente, não há nenhum ponto de referência completamente adequado para medir o processo de *software*. O rendimento do processo, a relação de custo da avaliação para o fracasso (A/FR) e a produtividade são úteis para o benchmarking do processo de *software*, mas eles não satisfazem completamente os critérios para um benchmark de propósito geral. Porém, eles são os melhores existentes e devem ser usados até que melhores medidas sejam inventadas.

O processo de *software* pode ser visto como a combinação de dois processos competidores: injeção de defeito e remoção de defeito. O conteúdo de defeitos do produto acabado é então governado pela diferença entre os resultados destes dois processos. Como ao fazer grandes alterações, mudanças relativamente pequenas em qualquer processo podem fazer uma diferença proporcionalmente grande no resultado final. Para administrar efetivamente a qualidade de *software*, é necessário focalizar ambos os processos – o de remoção e o de injeção.

Embora descobrir e corrigir defeitos seja extremamente importante, é uma estratégia inerentemente defensiva. Para fazer melhorias de qualidade significativas, deve-se identificar as causas destes defeitos e dar os passos para eliminá-los. As ações iniciais de prevenção de defeitos devem resolver a qualidade do projeto e do código e as ações que se podem tomar para assegurar-se que o processo definido é fielmente seguido.

4.6.3 PSP2 - Conteúdos do processo

O processo PSP2 introduz revisões de projeto e código. Ambos melhorarão a produtividade e a qualidade dos produtos. Com o PSP2, são iniciados os cálculos dos intervalos de predição para as estimativas de tamanho e tempo.

OBJETIVOS e PRÉ-REQUISITOS

Além dos objetivos para o PSP 1.1, os objetivos para o PSP2 são:

- introduzir revisões de projeto e código e
- introduzir métodos para avaliar e melhorar a qualidade das revisões.

As condições prévias para o PSP2 são o PSP1.l, a seção atual e as Listas de Revisão de Projeto e Código produzidas através da tarefa R4.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP2 - *Script* de Processo	**Tabela A-60**
PSP2 - *Script* de Planejamento	**Tabela A-61**
PSP2 - *Script* de Desenvolvimento	**Tabela A-62**
PSP2 - *Script* de *Postmortem*	**Tabela A-63**
PSP2 - Sumário do Plano de Projeto e Instruções	**Tabelas A-64 e A-65**
PSP2 - Lista de Revisão de Projeto	**Tabela A-66**
Lista de Revisão de Código	**Tabela A-67**
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-56 e A-57
Modelo de Planejamento de Horário e Instruções	Tabelas A-58 e A-59
Script de Estimativa PROBE	Tabelas A-45
Modelo de Relatório de Teste e Instruções	Tabelas A-46 e A-47
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-48 e A-49
Proposta de Melhoria de Processo (PIP) e Instruções	Tabelas A-37 e A-38
Padrão de Codificação	Tabelas A-13
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ESPECIFICAÇÕES DE RELATÓRIO DE PROCESSO

Os critérios de avaliação de processo do PSP2 incluem todos os itens de avaliação para o PSP1. Um quinto critério de avaliação para o PSP2 é que os dados históricos sejam consistentemente usados para melhorar o processo, por exemplo, para atualizar as listas de Revisão de Projeto e Código. Para satisfazer este critério, deve-se examinar os recentes dados de defeitos antes de cada projeto novo para ver se as listas de conferência devem ser modificadas. Nesse caso, as mudanças indicadas devem ser feitas e as listas de conferência novas usadas no próximo projeto. Anotar esta mudança na seção de notas e comentários dos PIPs submetidos com a tarefa e incluir as listas de conferência novas com o relatório de tarefa.

Os itens a serem incluídos nos relatórios de exercício do PSP2 e a ordem nas quais eles serão submetidos são:

- Sumário do Plano de Projeto do PSP2,
- Modelo de Relatório de Teste,
- Lista de Conferência de Revisão de Projeto do PSP2,
- Lista de Conferência de Revisão de Código,
- Os formulários PIP, inclusive com uma breve declaração do aprendizado nas lições,
- O Modelo de Estimativa de Tamanho,
- O Modelo de Planejamento de Tarefa,
- O Modelo de Planejamento de Horário,
- Log de Registro de Tempo,
- Log de Registro de Defeito,

- Listagem do Programa Fonte.

4.6.4 Exercícios

O plano de curso padrão para esta seção usa o PSP2 para escrever o programa 7A. Seguir fielmente o processo e o formato de submissão de relatório nele especificado.

Programa 7A

Programa 7A : condições prévias e referências: Relatório R4, programas 1A e 5A.

Programa 7A - requerimentos: escrever um programa para calcular a correlação entre duas séries de números e determinar a significação desta correlação. Usar a função de integração numérica do programa 5A para calcular o valor da distribuição e guardar os dados em uma lista encadeada.

Programa 7A - teste: testar completamente o programa. Como teste, usar os dados da Tabela A-20. Aqui os resultados para a correlação entre x e y devem ser r = 0.9543158, t = 9.0335, com 2*(1 - p) = 1.80*A10^{-5}. Esta é uma significação substancialmente melhor que (menos que) 0.005. Também usar o programa 7A para analisar os dados nos exercícios de programação até a data para determinar a correlação entre as LOC atuais novas e alteradas e o tempo de desenvolvimento atual, a correlação entre as LOC estimadas novas e alteradas e o tempo de desenvolvimento atual, e a significação destas correlações. Preparar e submeter um relatório de teste que inclua estes dados e usar o formato na Tabela A-21.

4.7 PSP2.1 – Modelos de Projeto

4.7.1 Projeto de *Software*

Muitos dos problemas históricos do desenvolvimento de *software* originaram-se de uma expectativa incorreta de que o desenvolvimento deveria começar com firmes e completos requerimentos. Porém, a história demonstra que, para um sistema de *software* novo, as exigências não serão completamente conhecidas até depois que o trabalho já tenha começado. Então, ao definir um processo de *software*, o engenheiro tem que reconhecer que os requerimentos provavelmente mudarão. Portanto, assim que possível, devem ser incluídos passos para identificar e solucionar incertezas de requerimentos.

Um das mais difíceis problemas de projeto refere-se ao *trade-off* entre a perfeição do projeto e os custos de desenvolvimento e cronograma. A especificação

completa de um projeto de *software* requer muita informação. Isto inclui definições para as classes e objetos que definem as suas relações, identificação dos dados trocados entre eles, definição dos dados requeridos e das transformações de estado e especificação das entradas e saídas do sistema. A completa e inambígua definição de todo este material geralmente requer uma quantia significativa de documentação.

À medida que o trabalho de projeto de *software* é feito, deve-se considerar duas diferentes questões: como o projeto é feito? Como o projeto tem que se parecer quando está terminado? O PSP não especifica um método particular de projeto. Porém, ele trata dos critérios de saída das fases de projeto. Para especificar corretamente a perfeição do projeto, as necessidades daqueles que usarão o projeto devem ser consideradas. Então, é possível determinar o que deve ser o produto do projeto.

Os pontos iniciais e finais do processo de projeto podem ser descritos com a ajuda de modelos. Estes modelos asseguram que os dados exigidos estão claramente definidos e concisamente representados. O Modelo de Especificação Funcional descreve as funções executadas por um programa, um objeto ou um procedimento. O Modelo de Especificação de Estado descreve o modelo de estado do programa. Uma completa especificação de estados contém uma descrição de cada estado de objeto e das transições entre eles. O Modelo de Especificação de Lógica usa pseudo-código para descrever precisamente a lógica de programa. O Modelo de Cenário Operacional[16] descreve o comportamento operacional do programa por um ou mais cenários (situações). Seu propósito é auxiliar a visualizar como o programa reage sob diversos cenários típicos do usuário.

Os modelos de projeto provêem, assim, uma base para o estabelecimento de critérios de saída de projeto. Os modelos podem ser usados para, progressivamente, refinar a especificação e o projeto de um *software*. Começando no nível mais alto, os modelos especificam as funções do programa principal, incluindo o seu relacionamento com o ambiente externo. Faz-se isso com os modelos de Especificação Funcional e Cenário Operacional. O comportamento interno de cada nível é definido com os Modelos de Especificação de Estado e de Lógica.

A figura abaixo mostra uma maneira de visualizar isso. Começando com uma declaração de requerimentos do programa, descreve-se as necessidades do usuário. A seguir esses requerimentos são traduzidos em uma especificação do programa. Como parte dessa especificação, pode-se completar um Modelo de Cenário Operacional e um Modelo de Especificação Funcional para o programa total. Este processo é repetido à cada nível sucessivo de projeto, completando um Modelo de Especificação de Estado e de Lógica para cada módulo ou objeto.

[16] *Operacional **Scenario** Template*

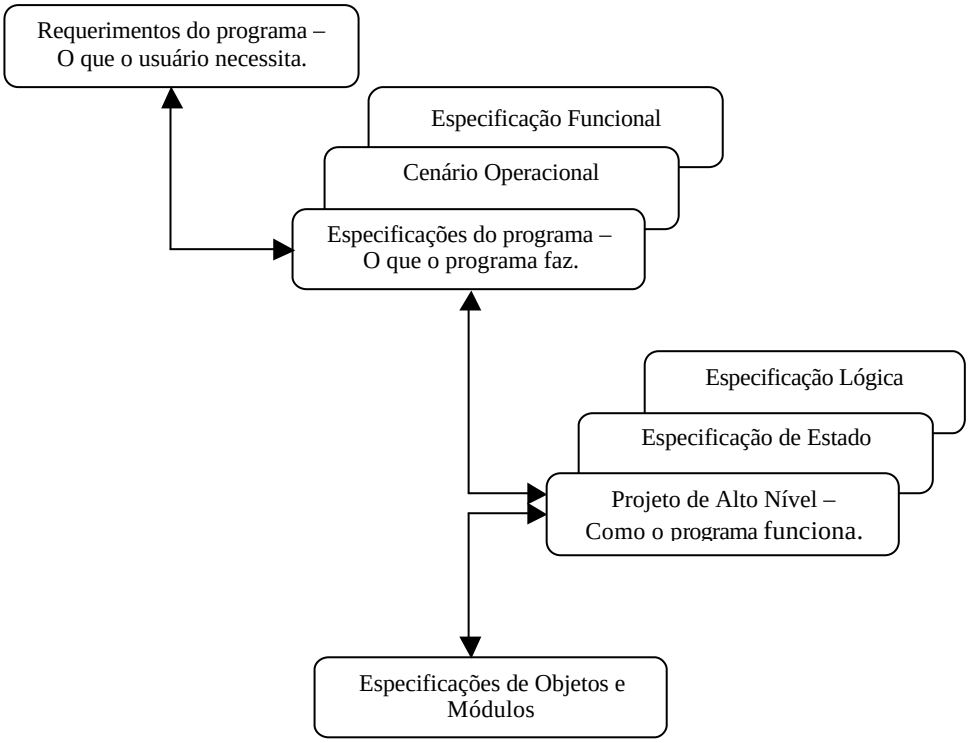

Figura 4-14 - Hierarquia de Projeto

No final, como mostra a Figura 4-14, , especifica-se o projeto detalhado para os módulos de programa de nível mais baixo.

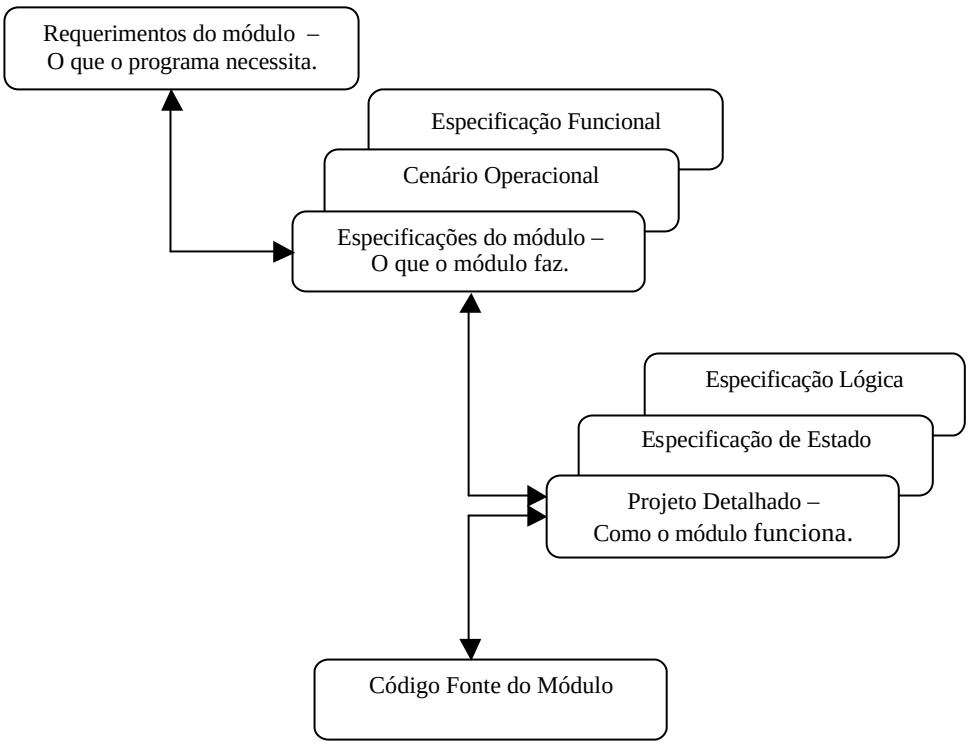

Figura 4-15 - Hierarquia de Implementação

Em essência, o projeto por abstração consiste de sucessivas desintegrações do sistema em partes manejáveis. A implementação, então, consiste em construir e integrar estas partes em um todo coerente. Durante o projeto, freqüentemente não é possível definir precisamente todos os subsistemas até que se saiba mais sobre eles. Algumas especificações de alto nível poderiam então ficar incompletas até que seja feito algum projeto de baixo nível. A seguir, o projeto poderia ser feito como uma série desconecta de projetos parciais. O engenheiro pode começar no topo e proceder até que encontre alguma abstração pouco conhecida. Se a abstração se parecesse particularmente difícil ou crítica, ela poderia ser projetada antes de proceder. Aquela abstração, por sua vez, pode invocar outras que se parecem igualmente críticas, e estas podem chamar outras. Como resultado, o engenheiro poderia se achar pesquisando profundamente no sistema antes de haver completado o projeto de alto nível.

Haverá casos nos quais não se poderá especificar um objeto externamente antes de projetá-lo completamente ou, possivelmente, até mesmo construí-lo e testá-lo. Nestes casos, protótipos destes objetos devem ser desenvolvidos antes de se completar os projetos de alto-nível. Depois de completados os protótipos e resolvidas as incertezas, os projetos de alto-nível podem ser completados.

4.7.2 PSP2.1 - Conteúdos do processo

O processo PSP2.1 introduz quatro modelos de projeto que provêem uma estrutura e formato para registrar os projetos. Embora eles não digam como fazer o projeto, podem ajudar a registrar o projeto corretamente quando estiver terminado.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Além dos objetivos para o PSP2, os objetivos para o PSP2.1 são:

- ajudar a reduzir o número de defeitos nos projetos,
- prover critérios para determinar se um projeto está completo, e
- prover uma estrutura consistente para verificar a qualidade dos projetos.

As condições prévias para o PSP2.1 são o PSP2 e a presente seção.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP2.1 *Script* de Processo	**Tabela A-68**
PSP2.1 - *Script* de Planejamento	**Tabela A-69**
PSP2.1 - *Script* de Desenvolvimento	**Tabela A-70**
PSP2.1 - *Script Postmortem*	**Tabela A-71**
PSP2.1 - Resumo de Plano de projeto e Instruções	**Tabelas A-72 e A73**
Modelo de Cenário Operacional e Instruções	**Tabelas A-75 e A-76**
Modelo de Especificação Funcional e Instruções	**Tabelas A-77 e A-78**
Modelo de Especificação de Estado e Instruções	**Tabelas A-79 e A-80**

Modelo de Especificação de Lógica e Instruções	**Tabelas A-81 e A-82**
Lista de Conferência de Revisão de Projeto do PSP2.1	**Tabela A-93**
Lista de Conferência de Revisão de Código	Tabela A-67
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-56 e A-57
Modelo de Planejamento de Horário e Instruções	Tabelas A-58 e A-59
Script de Estimativa PROBE	Tabelas A-45
Modelo de Relatório de Teste e Instruções	Tabelas A-46 e A-47
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-48 e A-49
Proposta de Melhoria de Processo (PIP) e Instruções	Tabelas A-37 e A-38
Padrão de Codificação	Tabelas A-13
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ELEMENTOS DE PROCESSO NOVOS

A única mudança na Lista de Conferência de Revisão de projeto do PSP2.1 é uma adição que se refere aos modelos de projeto introduzidos com o PSP2.1, descritos anteriormente.

ESPECIFICAÇÕES DE RELATÓRIO DE PROCESSO

Os critérios de avaliação de processo do PSP2.1 são os mesmos que aqueles para o PSP2, como segue:

- Os dados de processo estão completos.
- Os dados são precisos e autoconsistentes.
- O relatório de processo é submetido na própria ordem e formato.
- Os dados históricos são usados no planejamento do trabalho.
- São constantemente usados dados históricos para melhorar o processo.

Os itens a serem incluídos nos relatórios de exercícios do PSP2.1 e a ordem nas quais eles serão submetidos são como segue:

- **Resumo de Plano de Projeto do PSP2. 1**
- Modelo de Relatório de Teste
- **Lista de Conferência de Revisão de Projeto do PSP2.1**
- Lista de Conferência de Revisão de código
- O formulário PIP, inclusive com breve declaração das lições aprendidas
- O Modelo de Estimativa de Tamanho
- O Modelo de Planejamento de Tarefa
- O Modelo de Planejamento de Horário
- **Modelo de Cenário operacional**
- **Modelo de Especificação Funcional**
- **Modelo de Especificação de Estado**
- **Modelo de Especificação de Lógica**
- Log de Registro de Tempo
- Log de Registro de Defeito

- Listagem do Programa Fonte

4.7.3 Exercícios

São dados dois exercícios nesta seção, utilizando o PSP2.1 para escrever os programas 8A e 9A.

Programa 8A

Programa 8A - referências: programa 4A.

Programa 8A - requerimentos: escrever o programa 8A para ordenar uma lista encadeada de *n* números reais em ordem ascendente. Onde os itens de lista têm campos múltiplos, prover a capacidade para ordenar em qualquer campo. Assumir que a lista é curta o bastante para se ajustar na memória de computador.

Programa 8A - teste: testar completamente o programa. Como um teste, ordenar os dados nas duas colunas à direita da Tabela A-22. Fazer duas ordenações, uma em cada campo, e submeter um relatório de teste que descreva ambos os resultados. Nota: o programa não necessita imprimir os resultados.

Programa 9A

Programa 9A - Condições prévias: programas 5A e 8A
Programa 9A - requerimentos: escrever o programa 9A para calcular o grau para o qual uma seqüência de **n** números reais é distribuída normalmente. Usar o programa 5A para calcular os valores das distribuições normal e χ^2. Assumir que n é > 20 e mesmo um múltiplo de 5. Usar o programa 8A para ordenar os números em ordem ascendente.

Programa 9A - teste: testar completamente o programa. Como um teste, usar os dados de LOC/Método da Tabela A-22 como um caso de teste. Aqui, o resultado deve ser Q = 34.4 com um valor de probabilidade < 0.005 que os dados são distribuídos normalmente. Submeter um relatório de teste que inclua o resultado do teste e usar o formato da Tabela A-23.

4.8 PSP3 – Desenvolvimento Cíclico

4.8.1 Escalando o Processo Pessoal de *Software*

O tamanho dos projetos de *software* aumentou rapidamente durante mais de 30 anos. Assim, os produtos desenvolvidos no futuro serão provavelmente muito maiores dos que os de hoje. Como um processo que é ótimo para um programa pequeno não o será, provavelmente, para um processo cinco a dez vezes maior, provavelmente o processo de desenvolvimento terá que ser mudado.

Tipicamente, a fase de projeto de amplos sistemas de *software* começa com um esforço de projeto de sistema de alto nível, que divide o produto em componentes. Estes componentes são desenvolvidos separadamente e então o sistema é integrado. Como estes componentes são muito menores do que o sistema total, eles podem, presumivelmente, ser desenvolvidos com métodos de pequena escala.

Os tamanhos de projetos de desenvolvimento de *software* podem ser divididos em gamas gerais. Estes limites de escalabilidade são largamente determinados pela perícia e habilidade individual:

- na Fase[17] 0, se trabalha com os blocos de construção menores, como *loops*, *if-then-else*, *case*, etc.
- quando se combinam as construções da Fase 0 em um programa, move-se para a Fase 1. Na Fase 1, são desenvolvidos os módulos do programa. Estes programas ou módulos são típicos dos cursos iniciais de programação e tem um tamanho que variam de poucas dúzias até alguma centenas de LOCs. Como características dos processos dessa fase podem ser citados: não são escaláveis, usam métodos intuitivos que não possibilitam a construção das habilidades e disciplinas necessárias para o desenvolvimento de trabalhos em larga escala e, normalmente, são usados onde não se aplicam.
- Na Fase 2, são definidos os componentes de programas que contêm vários destes módulos. Agora, em vez de se focalizar nos detalhes de projeto, o desenvolvedor visualiza as interconexões destes módulos da Fase 1. Os programas são visualizados como abstrações.
- A Fase 3 trabalha com sistemas grandes, compostos de abstrações da Fase l e Fase 2, isto é, abstrai-se a essência, relegando grande parte da complexidade para os componentes de mais baixo nível. A complexidade do sistema é "mascarada", mantendo-se, porém, a qualidade dos componentes. Nesta fase, uma única pessoa não pode compreender os detalhes de todo o sistema.
- Sistemas de *software* verdadeiramente de larga escala na Fase 4 têm múltiplos subsistemas autônomos, cada qual manipulando responsabilidades do sistema inteiro.

[17] *Stage*

O PSP3 é um exemplo de um processo pessoal de larga escala. Ele pode apoiar razoavelmente desenvolvimentos individuais grandes ou pode servir como uma base para o desenvolvimento de *software* de larga escala. Sua estratégia é usar um processo cíclico. Uma estratégia de desenvolvimento bem fundada é construída sobre a estrutura natural do produto planejado. Esta estratégia define conteúdos de ciclo como elementos de tamanho pequeno que são projetados separadamente e implementados em ciclos de desenvolvimento. Cada ciclo é progressivamente testado e integrado e, ao fim, se tem o programa integrado e completo, pronto para a integração ou teste de sistema. O processo cíclico usado pelo PSP3 é mostrado na figura abaixo.

4.8.2 PSP3 - Conteúdos do processo

O PSP3 introduz um processo cíclico projetado para ajudar a desenvolver programas maiores. O fluxo conceitual do processo PSP3 é mostrado na Fig. 4-16. O processo começa com planejamento e projeto de alto-nível, seguido por uma série de ciclos de desenvolvimento. As principais preocupações da fase de projeto de alto-nível são produzir um projeto global e uma estratégia de desenvolvimento. Esta estratégia resolve testes, reusos e a estruturação de desenvolvimento de produto em incrementos que são, cada um, satisfatórios para o desenvolvimento com o PSP2.1 - como ciclos de processo.

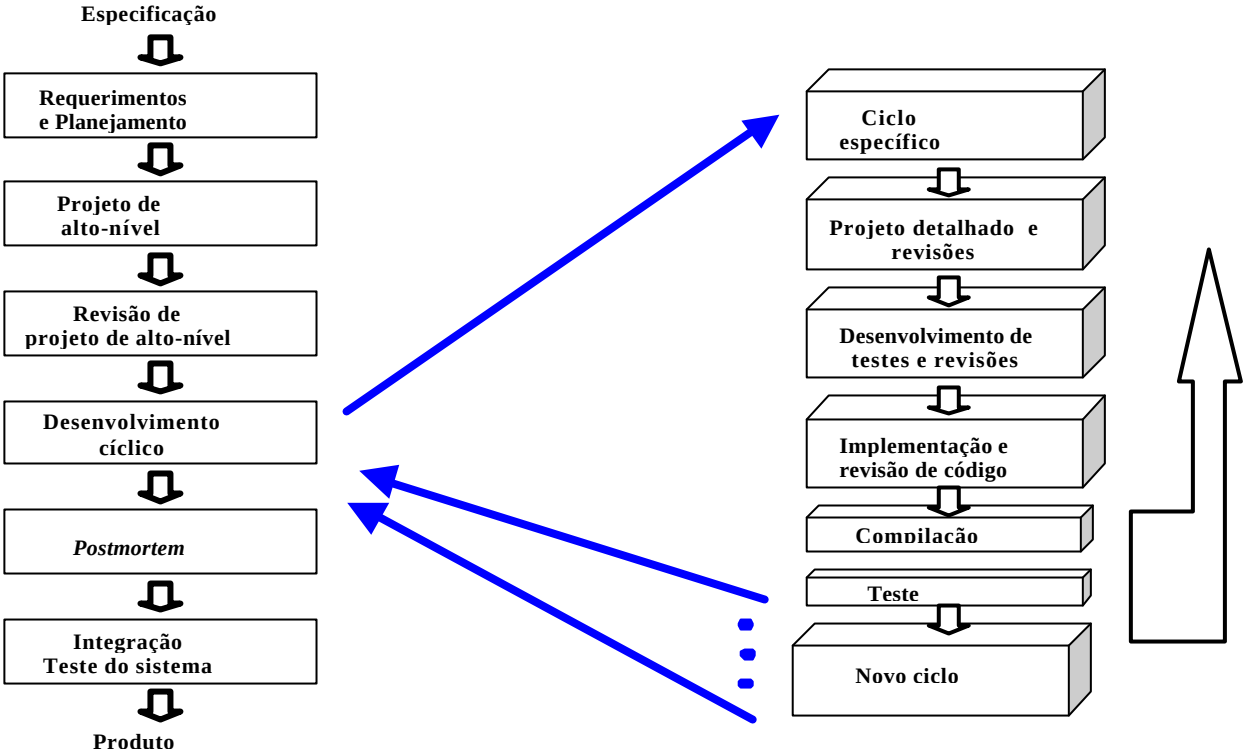

Figura 4-16 – O Processo PSP3

Requerimentos e Planejamento: produz um projeto conceitual do sistema como um todo, estima o tamanho e planeja o trabalho de desenvolvimento.

Projeto de alto nível: identifica a divisões naturais do produto e planeja uma estratégia cíclica. Cada ciclo deve ter entre 100 e 300 linhas de código fonte (novas e alteradas).

Desenvolvimento cíclico: primeiro, estabelece as especificações para o ciclo corrente. Cada ciclo é essencialmente um processo PSP2.1:

- cada ciclo é a base para o seguinte. Para ser preservada a escalabilidade, cada desenvolvimento incremental deve ser autocontido e livre de erros.
- Os testes devem ser desenvolvidos antes de se escrever o código. A vantagem de desenvolver e planejar testes antes é que isso força a pensar sobre o produto à partir de uma perspectiva de teste.
- Ao reavaliar o trabalho, determina-se o *status* atual. Caso necessário, pode-se fazer ajustes do plano.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Os objetivos do PSP3 incluem os objetivos do PSP2.1 mais um outro: estender a capacidade do processo pessoal para o desenvolvimento de programas de até várias mil LOC. A condição prévia para PSP3 é o PSP2.1.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP3 - *Script* de Processo	**Tabela A-83**
PSP3 - *Script* de Planejamento	**Tabela A-84**
PSP3 - *Script* de Projeto de Alto-nível	**Tabela A-85**
PSP3 - *Script* de Revisão de Projeto de Alto-nível	**Tabela A-86**
PSP3 - *Script* de Desenvolvimento	**Tabela A-87**
PSP3 - *Script* de *Postmortem*	**Tabela A-88**
PSP3 - Sumário do Plano de Projeto e Instruções	**Tabelas A-89 e A-90**
Resumo de Ciclo e Instruções	**Tabelas A-91 e A-92**
Log de Monitoramento de Assunto e Instruções	**Tabelas A-94 e A-95**
Modelo de Cenário Operacional e Instruções	Tabelas A-75 e A-76
Modelo de Especificação Funcional e Instruções	Tabelas A-77 e A-78
Modelo de Especificação de Estado e Instruções	Tabelas A-79 e A-80
Modelo de Especificação de Lógica e Instruções	Tabelas A-81 e A-82
Lista de Conferência de Revisão de Projeto do PSP3	Tabela A-93
Lista de Revisão de Código	Tabela A-67
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-56 e A-57
Modelo de Planejamento de Horário e Instruções	Tabelas A-58 e A-59
Script PROBE	Tabelas A-45
Modelo de Relatório de Teste e Instruções	Tabelas A-46 e A-47
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-48 e A-49
Proposta de Melhoria de Processo (PIP) e Instruções	Tabelas A-37 e A-38
Padrão de Codificação	Tabelas A-13
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

NOVOS ELEMENTOS DO PROCESSO PSP3

Além dos *scripts* do PSP3, os novos elementos de processo introduzidos com o PSP3 são o Log de Monitoramento de Assunto e o Resumo de Ciclo.

Log de Monitoramento de Assunto. O Log de Monitoramento de Assunto (ITL – *Issue Tracking Log*) provê um conveniente lugar para registrar assuntos, problemas e questões em aberto. No meio de um projeto, por exemplo, um nome de variável poderia ser alterado para representar melhor seu propósito. Em lugar de parar a cada ponto para se assegurar que cada ocorrência deste nome seja mudada ao longo do programa, este item poderia ser anotado no ITL para atenção posterior.

Resumo de Ciclo. Ao usar o PSP3, o desenvolvedor deve planejar implementar programas maiores em módulos de cerca de 100 LOC (esta cifra é arbitrária). Usa-se uma cópia do Resumo de Ciclo para registrar os planos de ciclo e outra para entrar com os resultados de ciclo atuais. Os dados do ciclo são fáceis de obter ao término de cada ciclo, mas são geralmente difíceis de reconstruir depois. No registro do tamanho, por exemplo, pode-se contar as LOCs totais do programa ao término de cada ciclo e determinar quais linhas foram reusadas, adicionadas, apagadas ou modificadas. Mais tarde, isto pode ser muito mais difícil.

É preciso salvar o tamanho planejado e atual, o tempo e os dados de defeito porque cada ciclo do PSP3 é essencialmente um pequeno projeto. Juntando estes dados e monitorando o desempenho pessoal, o desenvolvedor pode planejar e administrar melhor o trabalho. Ao considerar cada passo cíclico como uma tarefa e usar o método de valor merecido para monitorar o progresso do desenvolvimento, o engenheiro pode adquirir um quadro surpreendentemente preciso de seu estado e o cronograma de conclusão provável.

ESPECIFICAÇÕES DE RELATÓRIO DO PROCESSO

Os critérios de avaliação do processo PSP3 incluem todos aqueles para o PSP2.1, como segue:

- Os dados de processo estão completos,
- Os dados são precisos e autoconsistentes,
- O relatório de processo é submetido na própria ordem e formato,
- Os dados históricos são usados no planejamento do trabalho.
- Dados históricos são usados constantemente para melhorar o processo.

Em adição, o PSP3 tem um sexto critério: que dados históricos e a experiência sejam usados para estabelecer metas para a melhoria de processo no futuro. Derivativas em curto prazo desses objetivos devem ser usadas para estabelecer adequados objetivos desafiadores para cada projeto. Porém, somente estabelecer metas não seria adequado.

Para satisfazer estes critérios, é necessário identificar ações de processo para satisfazer as metas.

Os itens a serem incluídos nos relatórios do PSP3 e a ordem nas quais eles serão submetidos são:

Resumo de Plano de Projeto do PSP3
Resumo de Ciclo
Modelo de Relatório de Teste
Lista de Conferência de Revisão de Projeto do PSP3
Lista de Conferência de Revisão de Código
Log de Monitoramento de Assunto
Os formulários PIP, inclusive com breve declaração das lições aprendidas
O Modelo de Estimativa de Tamanho
O Modelo de Planejamento de Tarefa
O Modelo de Planejamento de Horário
Modelo de Cenário Operacional
Modelo de Especificação Funcional
Modelo de Especificação de Estado
Modelo de Especificação de Lógica
Log de Registro de Tempo
Log de Registro de Defeito
Listagem do Programa Fonte

Exercícios

A tarefa padrão para esta seção é usar o PSP3 para começar o desenvolvimento do programa 10A. Como este será, provavelmente, um programa maior que aqueles dos exercícios prévios, é planejado para dois períodos de tarefa. Ao completar esta tarefa, seguir fielmente o formato de submissão de relatório especificado para o PSP3.

4.8.2.1 Programa 10A

Programa 10A - condições prévias: programa 4A, 5A e 6A.

Programa 10A - requerimentos: escrever um programa para calcular os parâmetros de estimativa de regressão múltipla de três variáveis, fazer uma estimativa das entradas supridas pelo usuário, e determinar os intervalos de predição de 70 e 90 por cento. Mais adiante aumentar a lista encadeada do programa 4A.

Programa 10A - teste: testar completamente o programa. Como um teste, usar os dados da Tabela A-24. Para os valores estimados, usar 185 LOC de código novo, 150 LOC de código reusado e 45 LOC de código modificado. As horas projetadas devem ser de 20.76 horas, o UPI de 90 por cento é de 33.67 horas e o LPI de 90 por cento é de 7.84

horas. O intervalo de predição de 70 por cento é de 14.63 a 26.89 horas. O valor dos parâmetros beta são ß0 = 0.56645, ß1 = 0.06533, ß2 = 0.008719 e ß3 = 0.15105. Isto é equivalente a produtividade de 15.3 LOC novas e mudadas por hora, reuso de 114.7 LOC por hora, e modificação de 6.6 LOC por hora. Submeter um relatório de teste que dê os resultados no formato da Tabela A-25.

4.8.3 Verificação de projeto

Quando os projetos estão completos, claros e corretos, é possível produzir mais eficientemente uma implementação de qualidade. Os projetos são verificados para determinar se eles satisfazem os requerimentos e estão corretos. Além das especificações de programa, os padrões são uma parte importante de verificação. Os padrões pertinentes concernem às convenções de produto, projeto de produto e reuso.

Quando o produto incluir uma máquina de estado, deve-se verificar se a máquina é projetada corretamente e usada consistentemente. As condições para uma máquina de estado própria são que todas as condições de estado estejam completas e ortogonais, todas as transições de cada estado estejam completas e ortogonais e a máquina possa chegar a um estado de retorno de programa de cada estado. Também é importante examinar o contexto maior do programa para assegurar-se de que todos os estados de máquina[18] foram identificados e consistentemente usados.

Há vários modos para determinar a correção da lógica de um projeto antes de implementá-lo e testá-lo. Exemplos são tabelas de execução, tabelas de rastro e verificação matemática. Embora a verificação de tabela de rastro possa ser consumidora de tempo, ela não é mais do que identificar todos os casos necessários para um amplo teste de lógica. Se o engenheiro planeja desenvolver e rodar testes amplos, ele pode também capitalizar este esforço para verificar a correção do programa. Porém, se não tem a intenção de tais testes, então certamente ele deve fazer uma verificação de projeto muito completa.

Um corpo crescente da literatura lida com métodos matemáticos para provar a correção de programas. Com exceção das tabelas de execução, os métodos matemáticos formais são sofisticados e requerem considerável conhecimento e habilidade. Verificar a correção de projeto é simples em conceito, mas leva tempo para construir habilidades suficientes para se estar confiante de que as verificações estão completas e corretas.

[18] "state machines" no original.

Exercícios

A tarefa padrão para esta seção é usar o PSP3 para terminar o desenvolvimento do programa 10A. Este programa será provavelmente maior do que aqueles desenvolvidos previamente, assim ele foi planejado para dois períodos de tarefa.

4.8.4 Definindo o Processo de *Software*

O desenvolvimento do processo de *software* é muito parecido com o desenvolvimento de *software*. Tem artefatos semelhantes e requer muitas das mesmas disciplinas e métodos. Os passos para projetar um processo são: definir as necessidades, estabelecer metas e definir critérios de qualidade. A seguir, os processos atuais e projetados são caracterizados e é estabelecida uma estratégia de desenvolvimento. Finalmente, o processo inicial é definido e validado e são estabelecidos os meios para a sua melhora.

É praticamente impossível produzir um processo utilizável na primeira tentativa. O problema é que o trabalho feito para definir um processo muda o processo. Este conceito é melhor explicado pensando nos seguintes quatro processos: o processo percebido é o que se pensa que se faz, o processo atual é o que se faz de fato, o processo oficial é o que administração pensa que se está fazendo e o processo projetado é o processo para o qual se está evoluindo. Uma das razões principais pelas quais a qualidade dos processos é tão pobre é que freqüentemente não é feito o que se sabe que deve ser feito. Este ponto é importante porque não é possível evoluir o processo até que ele represente razoavelmente o que se está fazendo.

O desenvolvimento do processo deve ser planejado, medido, monitorado e melhorado. Diretrizes para um processo de desenvolvimento de processo efetivo incluem planejamento e medição do trabalho, monitoramento dos produtos produzidos e registro do tempo gasto por cada tipo de produto principal e categoria de desenvolvimento. Para planejar o desenvolvimento do processo, o engenheiro tem que definir sua medida de produtividade, usar estas medidas para planejar seu trabalho e manter um registro de desenvolvimento. Finalmente, ele deve produzir um relatório sumário para cada desenvolvimento de processo.

4.8.5 Exercícios

A tarefa padrão para esta seção requer que seja produzido um relatório final do processo do estudante do PSP. Este trabalho inclui três tarefas: atualizar o processo desenvolvido para o relatório R4, planejar o esforço requerido para produzir o relatório final e produzir o relatório.

R5 – Relatório Final

Avaliação: produzir um relatório sobre o que foi aprendido ao fazer os exercícios. A tarefa é proporcionar uma compreensão completa do desempenho atual do desenvolvimento de *software* do estudante do PSP e das áreas de maior prioridade para melhoria. Deve-se atualizar o processo usado para desenvolver o relatório central e usar este processo atualizado para produzir este relatório final.

Tarefas:

Tarefa 1: atualizar o processo desenvolvido para o relatório central. Submeter e atualizar cópia do processo e anotar as mudanças feitas e por que. Notar particularmente os PIPs que foram usados e que mudanças que foram feitas como resultado.

Tarefa 2: planejar e ordenar o processo definido na Tarefa 1, para fazer o trabalho definido na Tarefa 3. Usar o formulário de planejamento para registrar o tempo planejado para este trabalho e monitorar e registrar o tempo atual gasto. Submeter os dados atuais e planejados sobre o processo junto com o relatório final.

Tarefa 3: Analisar os dados para os programas desenvolvidos com o PSP. Análises através de planilha eletrônica são sugeridas e sumários gráficos e apresentações são encorajadas. No mínimo, produzir o seguinte:

- Analisar a acurácia da estimativa de tamanho e determinar o grau para o qual essas estimativas estavam dentro dos intervalos de predição estatísticos de 70 e 90 por cento. Também mostrar como a precisão de estimativa de tamanho evoluiu durante as tarefas.
- Analisar a precisão da estimativa de tempo e determinar o grau para o qual as estimativas estavam dentro dos intervalos de predição estatísticos de 70 e 90 por cento. Também mostrar como a precisão da estimativa de tempo evoluiu durante as tarefas.
- Analisar os tipos de defeitos injetados no projeto e codificação. Incluir uma análise de Pareto desses tipos de defeitos.
- Determinar as tendências para defeitos por KLOC achadas na revisão de projeto, revisão de código, compilação e teste. Também, mostrar as tendências para defeitos totais por KLOC ao longo do curso.
- Analisar a taxa de remoção de defeito para revisões de projeto, revisões de código, compilação e teste e mostrar a média de remoção de defeito para revisões de projeto, revisões de código e compilação *versus* teste de unidade. Naqueles casos nos quais não houve nenhum defeito de teste, usar a taxa média para remoção de defeitos de unidades de teste para os programas desenvolvidos até a data.
- Produzir uma análise do rendimento *versus* LOCs revisadas por hora nas revisões de código.
- Produzir uma análise de rendimento de revisão de projeto *versus* LOCs revisadas por hora.

Notar que o rendimento de revisão de projeto é calculado como segue:

Rendimento (DR) = 100*(defeitos removidos em revisão de projeto) / (defeitos removidos em revisão de projeto + defeitos que escaparam na revisão de projeto).

- Produzir uma análise do rendimento versus a taxa A/FR para os programas 7 até 10.
- Produzir um breve sumário descrevendo as áreas mais prioritárias para a melhoria de processo pessoal e por que. Resumir brevemente o desempenho atual, o desempenho futuro desejado e as metas de melhoria. Descrever como e quando se pretende satisfazer estas metas.

Resultado: Submeter as análises requeridas e um breve relatório escrito descrevendo as conclusões. Usar gráficos onde quer que seja possível. Notar particularmente como estes resultados serão usados para administrar e melhorar o *software* e outros trabalhos no futuro.

4.9 Conclusões

"PSP Limericks (by Anne Disney)
A programmer once went to school.
Thought to learn PSP - 'twould be cool.
But, oh!, all those forms,
Regression and norms...
She lit a fire with Humphrey as fuel.
(Note: "Humphrey" refers, of course, to the book, not the man!)"[19]

Em seu artigo *The personal Software Process In The Internet Age*, [RIC 00] manifesta-se de forma bastante otimista com relação ao PSP, qualificando-o como "o estado da arte na metodologia de engenharia de *software*". "Provavelmente o mais estruturado e medido processo ou curso de engenharia de *software* oferecido pelo SEI, livros, industria ou universidades". [RIC 00] Entretanto, como não poderia deixar de ser, o PSP tem qualidades **e** defeitos. Esta seção discute alguns prós e contras do PSP.

4.9.1 O custo do PSP

Aplicar o PSP a uma prática tem um custo significativo e um alto grau de compromisso.[HUM 95] Primeiro, leva tempo para se aprender e aplicar os seus métodos. **Aprender o PSP pode levar até quatro meses**. O trabalho de curso para o PSP tomará uma média de 130 horas. Depois de aprendidos os métodos do processo, ainda levará tempo para se colecionar e analisar os dados. À medida que melhorias de processo são implementadas, elas também levarão tempo para se acostumar. O tempo gasto ao se aplicar o PSP variará, dependendo das ferramentas usadas para aplicar os métodos. No caso de se usar caneta e papel, pode demorar um minuto para anotar cada entrada de tempo e cada defeito. Quando o projeto estiver acabado, pode levar uma hora para se reunir todos os dados e computar as métricas de desempenho.

É importante também a conclusão a que chegou [OLO 99a]: aprender os métodos do PSP toma muito tempo e não pode ser feito em paralelo com um curso de tempo integral.

Secundariamente, o PSP pode "desafiar" os sentimentos. Muitos esperam uma melhoria imediata do PSP, mas não funciona desta maneira. Enquanto que as pessoas que geralmente são novas no processo de *software* vêem a produtividade aumentada, o

[19] Extraído do site da University of Hawaii, http://csdl.ics.hawaii.edu/Research/PSP/PSPHumor.html, em 01/09/2000.

engenheiro com um método estável que aprende o PSP experimentará uma diminuição em sua produtividade [HAY 97].

Em sua tese, [COU 98] conclui que, embora o PSP tenha muitos benefícios, ainda há várias faltas que seu usuário terá que superar para receber esses benefícios. Em geral, não deve ser esperado que os usuários de PSP sigam os métodos exatamente como descritos. Ele salienta as vantagens do uso de formulários de controle de tempo, defeitos, o modelo da PIP, o Modelo de Relatório de Testes, planejamento e revisões. Entretanto questiona alguns erros ou faltas nesses formulários, o método de contagem de linhas de código lógicas como medida padrão de tamanho, o conceito de código reusado e a redundância que alguns modelos propostos pelo PSP poderão ocasionar no trabalho de engenheiros de *software* com práticas já estabelecidas, entre outros.

O PSP exige uma devoção disciplinada ao dever. Conforme [CAS 99] o processo de sete passos para a sua introdução "é uma série de ritos de iniciação crescentemente árduos, onde só os escolhidos pode sobreviver".

Finalmente, o PSP traz um risco de se estar em conflito com a auto-imagem. A pessoa tem que reconhecer as suas forças e nunca deve enfatizar as suas fraquezas.

4.9.2 Os benefícios do PSP

Embora o PSP tenha vários custos, Humphrey argumenta que um número grande de benefícios pode ser percebido quando o processo pessoal da pessoa foi estabilizado [HUM 95a]. Eles são listados a seguir:

- O programador ganha uma avaliação das suas forças e fraquezas, mostrada nos dados do PSP.
- O programador pode interpolar dados colecionados de tal forma que podem ser derivadas idéias para a melhoria de processos.
- O programador pode resistir a uma pressão irracional por discussão do tamanho antecipado do problema e relacionar isto a sua produtividade histórica.
- A conclusão organizada de um projeto fornece ao programador um senso de realização pessoal.
- Considerando que o programador tem um processo repetível, consistente e estável, ele vai ganhar mais confiança do seu grupo.

Benefícios do PSP para a Organização

O PSP introduz os seguintes benefícios para a organização:

- Dados do PSP melhoram o planejamento e o gerenciamento do cronograma de projetos de *software*.
- A remoção de defeitos introduzidos antes da compilação e testes resulta em um produto de melhor qualidade, reduz o custo dos testes e diminui o tempo de desenvolvimento.
- O PSP introduz um aprendizado e prática para a melhoria do processo. Pequenos ciclos de desenvolvimento e os dados pessoais tornam fácil o entendimento e aumenta a experiência.
- O PSP ajuda os engenheiros e seus gerentes a aprenderem a prática da quantificação do gerenciamento do processo. Eles aprendem o uso do processo definido e aprendem também a coletar dados para gerenciar, controlar e melhorar o trabalho.
- Finalmente, o PSP expõe os engenheiros a 12 áreas chaves do CMM (KPAs). Com isso facilita a preparação dos engenheiros a participar na melhoria baseada no CMM.

4.9.3 Revisão de projeto e código

Uma questão em aberto é se o código deve ser revisado antes ou depois da compilação. Humphrey argumenta que cerca de 10% dos erros de sintaxe não são descobertos pelo compilador. Apesar de dizer que nada impede que a revisão seja feita após a compilação, manifesta-se totalmente favorável à revisão antecipada. Talvez aqui a questão esteja mais no modelo de construção do produto do que em questões específicas. Na realidade, Humphrey segue um raciocínio linear (em cascata) de desenvolvimento: primeiro planejamento, depois projeto, etc. A questão é: esse modelo é válido? O autor deste trabalho utiliza uma abordagem mista ao desenvolver programas: apesar de planejar linearmente o produto, o desenvolvimento é efetuado de forma cíclica, com o uso freqüente de protótipos. Essa abordagem não se adapta confortavelmente aos métodos de revisão proposto. Além disso, fazer manualmente um trabalho que pode ser feito pelo compilador de forma bastante eficiente não parece sensato. Acrescente-se a isso todo o trabalho requerido pelos controles, *scripts*, planilhas e modelos propostos pelo PSP e já se terá argumentos suficientes a favor de efetuar a compilação antes de revisar o código.

A visão adversária [Crawford apud [CAS 99]] é que as ferramentas são mais custo-efetivas na detecção de erros de sintaxe e que o desenvolvedor pode aprender tanto ou mais com as estatísticas de erro geradas pelo compilador do que ao achar ele próprio os erros de sintaxe. Segundo Casey, os argumentos de Humphrey não são

convincentes. [CAS 99] Ele se pergunta: pode uma revisão ser mais efetiva na localização de defeitos complexos se é gasto tempo corrigindo erros de sintaxe triviais?

4.9.4 Automação: sim ou não?

Considera-se que uma das principais questões sobre o PSP refere-se à sua automação. Embora Humphrey não se mostre favorável ao uso de ferramentas totalmente automatizadas[20], é preciso atentar para o grande esforço exigido para completar e controlar todos os formulários, *scripts* e modelos propostos. Além disso, [COU 98] salienta quatro desvantagens importantes na abordagem manual do PSP:

- Usar lápis e papel para executar o PSP é tedioso.
- O ser humano é naturalmente propenso a erro.
- Formulários prontos de papel não permitem muito ao usuário divergir do processo prescrito.
- Uma correção em um formulário forçará o recálculo de muitos campos, que levam muito tempo à mão.

A questão é: a Engenharia de *Software* deve municiar o desenvolvedor com ferramentas e técnicas efetivas, que na prática signifiquem uma melhora na atividade de desenvolvimento, ou deve o desenvolvedor despender uma quantia significativa de seu tempo com o manuseio de um volume considerável de papéis? Acredita-se na efetividade da primeira questão.

Porém, é preciso atentar para o fato que uma ferramenta mal desenvolvida seria uma inconveniência a seus usuários. Uma ferramenta de apoio do PSP deve ser transparente com respeito à manipulação de tarefas, tal que os engenheiros possam se **focalizar nos seus produtos em vez de nos seus processos**. [COU 98] e [WIL 97] No capítulo 5 são relacionadas algumas ferramentas disponíveis para o PSP.

Indo mais além, sugere-se a possibilidade de implementação de uma ferramenta de apoio do PSP **incorporada ao compilador** do desenvolvedor ou ao ambiente ou editor de programação, de tal forma que simplifique o trabalho sem diminuir sua qualidade.

Essa sugestão já foi criticada por [OLO 99], sob a alegação de que tal ferramenta desencorajaria o uso das revisões de projeto e código. Entretanto, essa alegação é rebatida com um exemplo: os atuais SGBDs são ferramentas extremamente sofisticadas, e muitas das atividades anteriormente de responsabilidade dos desenvolvedores de bancos de dados agora são feitas de forma automática. Mas, sem um bom conhecimento e uso de teoria de bancos de dados, é virtualmente impossível desenvolver um banco de dados. Da mesma forma, acredita o autor, acontecerá com o PSP.

---

[20] O Professor Jones Oliveira de Albuquerque (joa@comp.ufla.br), em *e-mail*, informa que, em conversa com o próprio Humphrey constatou que ele (Humphrey) não gosta da idéia da automatização, opinião já evidenciada no livro.

Além disso, Humphrey deixa bem claro que o PSP foi projetado para aqueles desenvolvedores que querem fazer um trabalho de qualidade, seguindo um processo definido ao desenvolver programas. O uso de facilidades apenas incentivará o cumprimento das tarefas propostas.

Integrado dentro de um ambiente de programação ou editor, como Borland C++ Builder ou Delphi, o editor e o programa de PSP poderiam trocar informação automaticamente, ao compilar defeitos, por exemplo. Também poderia ajudar a contar LOCs, desde que o programa pudesse estar atento às mudanças feitas pelo usuário em tempo real. Obviamente, aqueles dados que não pudessem ser obtidos de forma automática, seriam fornecidos pelo desenvolvedor.

5 Estudos do Impacto do PSP

> "The challenge over the next 20 years will not be speed or cost or performance; it will be a question of complexity."
> Bill Raduchel, Chief Strategy Officer, Sun Microsystems

O Impacto do PSP

O PSP é uma tecnologia relativamente nova, porém está causando impacto. [COU 98] Aqueles que fizeram o curso de PSP estão notando melhorias. Além disso, seus métodos estão sendo aplicados na indústria e em estudos de universitários.

5.1 O Estudo do SEI

Em 1997 Will Hayes e James W. Over fizeram um estudo com 298 engenheiros de *software* e o resultado deste estudo está disponível no relatório técnico "*An Empirical Study of Impact of PSP on Individual Engineers*" disponível no site do SEI.[HAY 97]

O relatório descreve o efeito de PSP em dimensões de desempenho fundamentais destes engenheiros, incluindo a sua habilidade para calcular e planejar o trabalho, a qualidade do *software* que eles produziram, a qualidade do seu processo de trabalho e a sua produtividade. O relatório também discute como melhorias na capacidade pessoal também melhora o desempenho organizacional em várias áreas: custo e administração de horários, qualidade de produto entregue e tempo de ciclo de produto.[HAY 97]

Resultados do estudo:

No estudo foram examinadas cinco dimensões de melhoria de processo pessoais do PSP: **acurácia da estimativa de tamanho e esforço, qualidade do produto, qualidade do processo e produtividade pessoal**. Foi concluído que o PSP melhorou o desempenho nas primeiras quatro dimensões sem qualquer perda na quinta área, produtividade.

Estimativa de tamanho

Quando o método de estimativa de tamanho é introduzido no começo do PSP1, os engenheiros têm dados das três tarefas prévias como uma base para calcular o quarto. Então, a hipótese testada nesta seção do estudo é:

"À medida que os engenheiros progridem através do treinamento do PSP, suas estimativas de tamanho gradualmente aproximam-se do tamanho atual do programa finalizado. Mais especificamente, com a introdução de uma técnica de estimativa formal de tamanho no PSP nível 1, existe uma melhoria notável na acurácia das estimativas de tamanho dos engenheiros."

Conclusão: o estudo observou uma melhora da ordem de 150% (em média) na estimativa de tamanho.

Estimativa de esforço

O PSP ajuda os engenheiros, neste caso, armando-os com dados de estimativa de esforço. Nesta seção, foram usados dados para testar a seguinte hipótese:

"À medida que os engenheiros progridem através do treinamento do PSP, as suas estimativas de esforço aproximam-se ao esforço atual gasto durante o ciclo de vida inteiro. Mais especificamente, com a introdução de uma técnica estatística (regressão linear) no nível de PSP1, há uma melhoria notável na acurácia das estimativas de esforço dos engenheiros".

Conclusão: o estudo observou uma melhoria de 75% (em média) na estimativa de esforço.

Além disso, a tendência de subestimar o tamanho e o esforço foi reduzida. E o número de sobreestimativas e de subestimativas foi mais balanceado.

Densidade de defeitos

O PSP usa este método para medir a **qualidade do produto**, como também várias métricas de qualidade de processo. As hipóteses a ser investigadas nesta seção foram:

"À medida que os engenheiros progridem através do treinamento do PSP, o número de defeitos injetados e então removidos por mil linhas de código (KLOC) diminui.
Com a introdução de revisões de projeto e código no PSP nível 2, a densidade de defeitos de programas que entram nas fases de compilação e teste diminui significativamente".

Conclusão: os defeitos encontrados em unidade de produto testada melhoraram em uma relação de 2.5 vezes (em média).

Rendimento de defeitos antes da compilação

Rendimento é a porcentagem dos defeitos injetados antes da fase de compilação que são removidos antes da primeira compilação. O PSP ensina aos engenheiros a

examinar a **qualidade do processo** quantificando o seu rendimento do processo de *software* pessoal. A hipótese a ser testada nesta seção foi:

"À medida que os engenheiros progridem através do treinamento do PSP, seu rendimento aumenta significativamente. Mais especificamente, a introdução de revisão de projeto e código no PSP1 tem um impacto significante no valor de rendimento dos engenheiros".

Conclusão: os defeitos encontrados antes da compilação aumentaram em 50% (em média).

Produtividade

A hipótese a ser testada nesta seção foi:

"À medida que os engenheiros progridem através do treinamento do PSP, a sua produtividade aumenta. Isto é, o número de linhas de código projetadas, escritas e testadas, aumenta entre as primeiras e últimas tarefas".

Conclusão: o número de linhas de código por hora não alterou substancialmente. Mas a melhoria da qualidade do produto resulta na melhora da produtividade do ciclo de desenvolvimento. Testes de produto e testes de integração são executados mais rapidamente por causa da melhoria da qualidade do produto, diminuindo assim o ciclo de desenvolvimento.

Conclusões gerais do estudo:

Os objetivos do estudo eram examinar o efeito do Processo de *Software* Pessoal no desempenho de engenheiros de *software* e considerar se os resultados observados poderiam ser generalizados além dos participantes de estudo. Como o PSP foi desenvolvido para melhorar o desempenho individual pela introdução gradual de práticas novas, o estudo tomou uma abordagem semelhante, examinando a mudança no desempenho individual à medida que estas práticas foram introduzidas.

Foi concluído que as melhorias em quatro dimensões, precisão de estimativa de tamanho, precisão de estimativa de esforço, qualidade de produto e qualidade de processo, eram estatisticamente significativas. Nenhuma mudança estatisticamente significativa na produtividade foi achada. Assim, o estudo declarou que as melhorias observadas nas outras dimensões de desempenho foram alcançadas sem qualquer perda de produtividade.

Concluindo, as análises informadas no relatório substanciam que as tendências no desempenho pessoal, observadas durante o treinamento do PSP, são significativas, e que as melhorias observadas representam real mudança no desempenho individual, não uma mudança no desempenho comum do grupo. Além disso, concluiu-se que as melhorias observadas ocorreram devido ao PSP e podem ser generalizadas.

5.2 A análise de Humphrey

Humphrey também escreveu um relatório sobre melhorias notadas no curso de PSP. A análise dele cobriu 104 estudantes em oito grupos. Humphrey notou os seguintes impactos: [COU 98]

- Uma melhoria significativa na densidade de defeito ($p < 0.005$), apoiando as conclusões de Hayes e Over.
- O rendimento de defeito melhorou significativamente à medida que os engenheiros procederam pelo tempo de curso. Como com Hayes e Over, nenhuma melhoria significativa foi feita até que as revisões de código e projeto foram introduzidas no PSP2.
- Humphrey concluiu que os programadores sem experiência tiveram uma melhoria na sua produtividade, à medida que as suas taxas de defeito eram reduzidas. Semelhantemente, os programadores experientes mostraram um declínio na produtividade, como eles tiveram que adicionar o *overhead* extra de fazer as tarefas requeridas pelo PSP. Humphrey concluiu que maximizar a produtividade não é em si mesmo um benefício para um engenheiro. Enquanto que as tarefas do PSP levam tempo, o desempenho apropriado das tarefas do PSP conduz a produtos de qualidade mais altos e mais baixos tempos de teste.

5.3 ESSI

O *European System & Software Initiative* (ESSI), em conjunto com outras instituições européias, tem promovido diversos experimentos para verificar a eficiência e eficácia do PSP. Alguns dos experimentos são:

- PIBOP (*Process Improvement Based on PSP*): O objetivo da experiência foi introduzir a abordagem do PSP no Centro de Desenvolvimento de *Software* de Intracom S.A., empresa grega.[KOP 99]

- PERSPI (*PERsonal Software Process Improvement*): o PERSPI é uma tentativa de introduzir a metodologia de PSP em um ambiente industrial, pondo ênfase em seu emprego gradual em projetos da vida real, em lugar de em uma introdução por treinamento formal e a longo prazo. A empresa escolhida para o experimento foi a Computer Logic S.A., também uma empresa grega. [MEN 98]

- ASPIDE (*Application of a Strategy of Personal Planning to Improve the Development Engineering*): o projeto ASPIDE é a aplicação do Processo de *Software* Pessoal, passos PSP0 e PSP0.1, aos empregados de Socrate Sistemi S.a.S., empresa italiana. [GUI 99]

6 Tópicos atuais sobre o PSP

> "A atenção deve ser focalizada em educar uma nova geração de Engenheiros de *Software* com processos e princípios que vão resolver a Crise do *Software* e restabelecer a integridade da indústria de *software*." [WIL 97]

6.1 PSP na educação

Existem cerca de 20 Universidades que estão oferecendo o PSP nos seus currículos acadêmicos de graduação. [RIC 00] Essas universidades acreditam que usando os métodos do PSP, permitiriam aos estudantes desenvolverem uma base para o desenvolvimento de *software* disciplinado que está sendo buscado pela indústria [HIL 97]. Hilburn acredita que as práticas de PSP precisam ser introduzidas no começo do programa, caso contrário os estudantes desenvolveriam hábitos pobres de programação. [COU 98]

Conceitos como administração de tempo podem ser introduzidos no primeiro ano e podem ser continuados ao longo do programa. O registro e a prevenção de defeitos também seriam cobertos. Os métodos causaram uma reação diversa por parte de estudantes. Conforme relata Dale Couprie, estudantes reconheceram que os conceitos os beneficiariam, tanto no nível acadêmico como profissional. Porém, eles não gostaram de registrar os contínuos tipos de defeitos comuns, como erros de digitação. [COU 98]

Um dos exemplos da aplicação do PSP em cursos universitários vêm da *Texas Tech University*. Naquela universidade, oito cursos relacionados no currículo de informática evoluíram para um currículo integrado de engenharia de *software*. Os cursos mais coesos nesta sucessão são três cursos de nível junior/senior: *"Software Engineering", "Senior Project Design", e "Senior Project Implementation Laboratory"*. Cerca de metade do curso *"Senior Project Design"* envolve o estudo do PSP. As técnicas aprendidas nesse curso são objetivamente aplicadas no curso seguinte, *"Senior Project Implementation Laboratory"*. [BAG 97]

Outro exemplo bastante interessante é o trabalho desenvolvido na *University of Utah*. Esta universidade vem fazendo um trabalho permanente de adaptação dos estudantes ao PSP para melhorar a participação destes, a retenção e a utilização dos materiais e conceitos do PSP.[WIL 97]

A seguir, são listadas outras instituições que adotaram o PSP:

Através de uma iniciativa chamada "Fazendo Trabalho de Qualidade", a *Embry-Riddle Aeronautical University* está introduzindo material sobre processos nos dois primeiros cursos de ciência da computação (http://erau.db.erau.edu/~psp/info/). Muitas das noções encontradas no PSP estão incluídas na abordagem deles. [UMD 97]

PSP Resources Page da *University of Karlsruhe***:** a Universidade alemã mantêm um site com endereços de recursos para suportar o PSP (http://wwwipd.ira.uka.de/PSP/). O IPD *Software Engineering Group*, desta Universidade, usa e ensina o PSP e oferece ferramentas de PSP para usuários em geral.

O Departamento de Ciências da Computação e Informação da *East Tennessee State University* desenvolveu o *PSP Studio* e mantém um site para download do programa e manuais (http://www-cs.etsu.edu/psp/).

Desde seu começo, o Centro de Engenharia de *Software* – CSE - da *Dublin City University* identificou a melhoria do processo de *software* (SPI) como uma área de prioridade. Esta universidade é o foco nacional designado para informação e ajuda em qualidade de *software* e melhoria de processo dentro da infra-estrutura de apoio estatal irlandesa. O CSE já efetuou, entre outras iniciativas, o primeiro treinamento público disponível na Europa sobre o Processo de *Software* Pessoal (PSP).

Existem vários relatórios sobre o PSP na página de Publicações da *University of Hawaii* (http://csdl.ics.hawaii.edu/Publications/Publications.html). A Universidade é bastante referenciada em páginas de PSP na Internet.

A *Universidade Federal de Minas Gerais* mantêm um *site* sobre o PSP (http://www.dcc.ufmg.br/~joa/opcoes_menu/cursos/aula_psp/index.html) e, no segundo semestre de 1998, incluiu no currículo do Departamento de Ciências da Computação a disciplina experimental Tópicos em Ciência da Computação: Processo Pessoal de *Software*.

A Pós-Graduação em Engenharia de *Software* do *SENAC-SP* oferece uma disciplina de PSP (http://www.sp.senac.br/). Aquele SENAC oferece, ainda, assessoria para adoção de modelos de garantia de qualidade em *software*, entre eles, o *Personal Software Process* (PSP), *Team Software Process* (TSP) e *Capability Maturity Model* (CMM).

Como parte da IV Semana de Engenharia de *Software*, O SENAC-SP ofereceu o curso de "Introdução à Melhoria de Processo de *Software*" nos dias 23 e 24 de agosto/2000. Este curso foi ministrado pelo *European Software Institute* - conforme o SENAC um dos maiores centros dedicados à melhoria de processo de desenvolvimento de *software* (vide abaixo). O curso abordou, entre outros temas, os modelos SEI CMM e SEI PSP.

O *European Software Institute* (ESI) é uma das autoridades na melhoria do processo de *software*. Estabelecido em 1993 e com sua sede na Espanha, o ESI é uma organização não-lucrativa dirigida às demandas da indústria européia. É apoiado pela

Comissão Européia, pelo governo basco e por sócios. Promove diversos cursos, seminários e palestras, muitos deles sobre o PSP (http://www.esi.es). É um dos parceiros do ESSI no projeto TRAPSP, descrito abaixo.

O Projeto TRAPSP

O propósito principal de projeto TRAPSP (*Advanced Multimedia and Distance Learning for SME Teams using the PSP*) é desenvolver uma aplicação baseada em técnicas de multimídia e de aprendizagem à distância, para treinar engenheiros de *software* e gerentes seniores no modelo PSP. [SAN 97]

O objetivo da aplicação TRAPSP é treinar os usuários finais da aplicação nos níveis PSP0 e PSP1.

O Projeto TRAPSP é patrocinado pelo programa ESSI da Comunidade Européia. As organizações participantes são o *LABEIN Technological Research Centre*, da Espanha, como coordenador do projeto e o ESI como subcontratante, entre outras.

O Consórcio é co-operacional com outras quatro Experiências de Melhoria de Processo (*Process Improvement Experiencies* – PIE) e outros projetos do grupo de treinamento do ESSI, fundados pela Comissão Européia.

O TRAPSP começou em primeiro de março de 1997.

6.2 PSP na indústria

Segundo o SEI, a primeira organização a informar seus resultados após usar o PSP e o TSP foi a Teradyne. Sua análise de retorno de investimento (*return on investment* - ROI) indica que, usando o PSP e o TSP em 2 projetos que juntos somam 112 KLOC, eles foram beneficiados em aproximadamente $5.3 milhões de dólares até a data, em tempo de engenharia economizado.[SEI 00a]

Com código desenvolvido usando o TSP/PSP, eles estão economizando aproximadamente 120 hours/KLOC em integração de sistema e testes. Os níveis de qualidade melhoraram 20 vezes sobre projetos anteriores e o esforço atual e horários estavam dentro de 8% de planejado. A Teradyne estima que o custo de treinamento do PSP é de um mês por engenheiro.

A Boeing, a AIS - Advanced Information Services, Inc. e a Base da Força Aérea de Colina (EUA), também informaram benefícios significativos recebidos pelo uso do PSP e TSP. Os quadros seguintes (Figuras 5-1 até 5-5) são uma compilação de dados de resultados da Teradyne, Boeing, AIS e AFB-Colina.[SEI 00a] Dados de 18 projetos de TSP/PSP nestas organizações foram combinados para ilustrar a variação de desempenho (antes e depois) para:

- *Post-release defects/KLOC*

- Duração de testes de sistema
- *Defects/KLOC* em testes de aceitação
- Desvio de horários
- Desvio de esforços

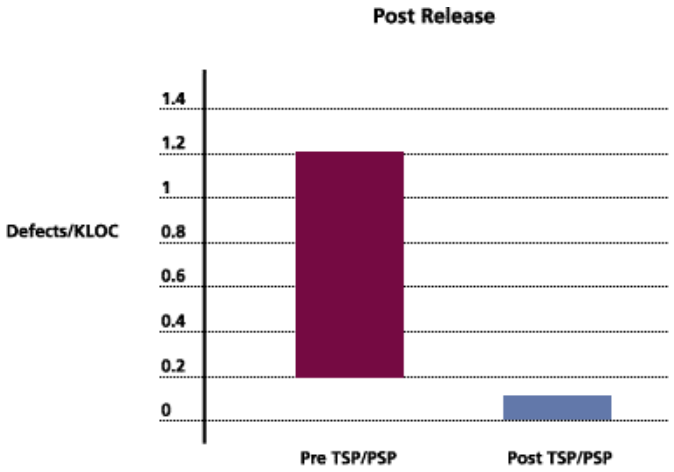

Figura 6-1 - Defeitos/KLOC

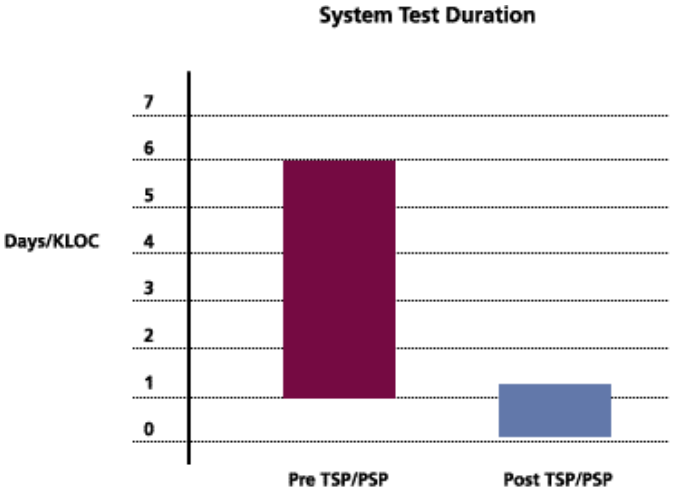

Figura 6-2 - Duração de testes de sistema

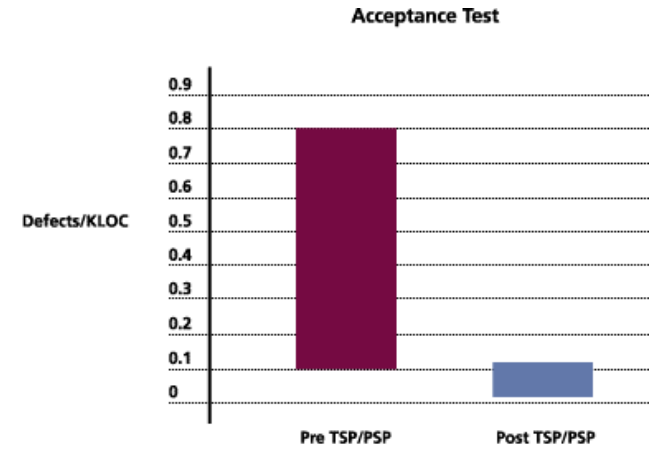

Figura 6-3 - Defects/KLOC em testes de aceitação

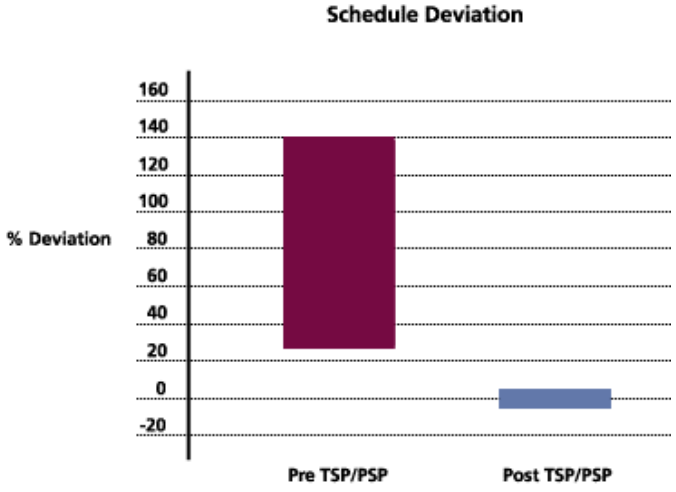

Figura 6-4 - Desvio de horários

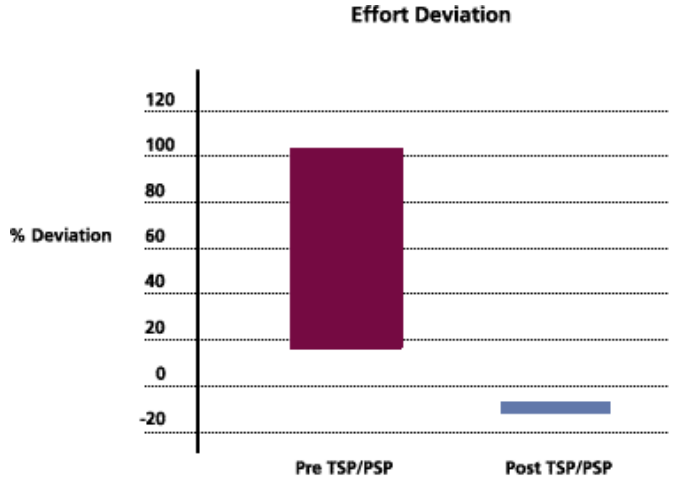

Figura 6-5 - Desvio de esforços

Particularmente, com relação à Boeing Company, foram observados os efeitos mais notáveis ao acrescentar o PSP ao Programa de Melhoria de Processo da empresa:

- O PSP foi introduzido na Liberação (Release) #9,
- Embora fosse 2.36 vezes maior que as três liberações anteriores,
- Havia 75% menos defeitos descobertos durante os testes de sistema,
- Resultando em 94% de economia em tempo de teste (vide figuras abaixo).

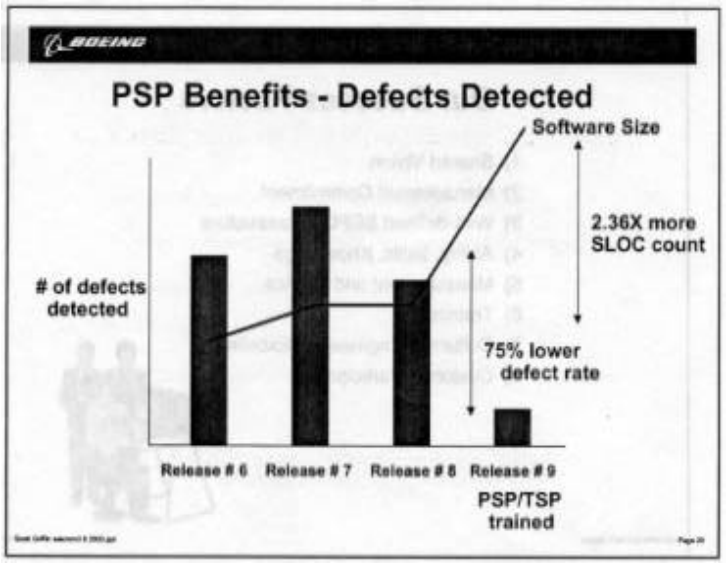

Figura 6-6 - Benefícios do PSP - Defeitos Detectados

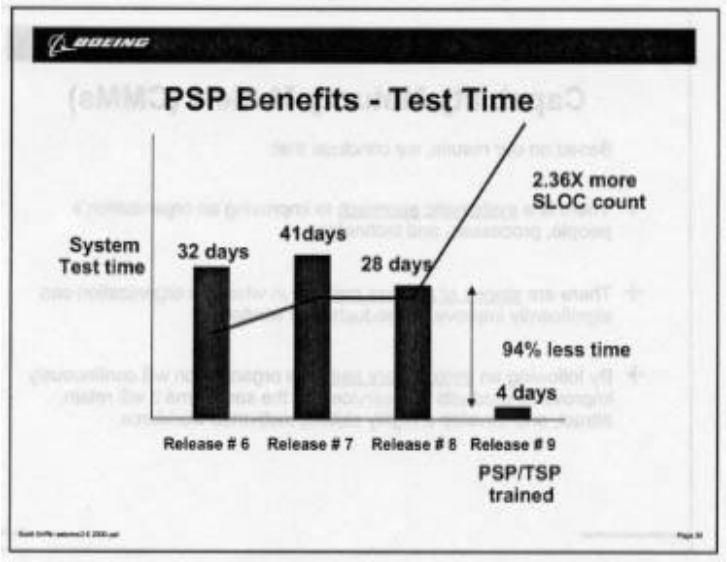

Figura 6-7 - Benefícios do PSP - Tempo de Teste

Os dados do uso industrial sobre os efeitos do PSP são limitados, devido aos seguintes fatores: [KOC 00]

- Muitas companhias são relutantes em liberar dados específicos que competidores ou clientes poderiam usar para identificar custos atuais e níveis de defeito.
- A maioria das companhias tem poucos ou nenhum dado histórico para comparar com dados do PSP; assim é difícil de quantificar os efeitos do PSP.
- O PSP ainda não foi adotado amplamente (como o CMM); assim há menos casos disponíveis.

Obviamente, esses dados referem-se aos EUA. No Brasil, praticamente inexistem empresas usando o PSP.

6.3 Suporte automatizado ao PSP

A execução do PSP a caneta e papel envolve um compromisso grande de tempo e está sujeita a erros. Na fase de *Postmortem* - a fase depois dos testes – algo em torno de 30 minutos a uma hora pode ser consumido para calcular as estatísticas e completar os formulários. Este compromisso de tempo pode desencorajar as pessoas a usar o processo.

Dale Couprie salienta quatro desvantagens importantes na abordagem manual do PSP: [COU 98]

- Usar lápis e papel para executar o PSP é tedioso.
- O ser humano é naturalmente propenso a erro.
- Formulários de papel não permitem muito ao usuário divergir do processo prescrito.
- Uma correção em um formulário forçará o recálculo de muitos campos, que levam muito tempo à mão.

Estas desvantagens podem ser aliviadas se o processo for automatizado. Pelo uso de uma ferramenta de apoio de PSP podem ser percebidos os seguintes benefícios:

- A coleta de dados de tempo e defeito seria simplificada.
- Os cálculos requeridos pelo PSP são computados rapidamente.
- Os dados dos formulários podem ser acessados instantaneamente.

- Seria desnecessário duplicar dados, fazendo dados consistentes entre formulários.
- Dados históricos seriam armazenados convenientemente para análise.
- Uma orientação automatizada na sucessão de tarefas poderia ser seguida facilmente.
- Enganos podem ser corrigidos facilmente através de facilidades de edição.
- A precisão de dados do projeto é limitada ao poder de computação da plataforma.
- Um usuário poderia facilmente personalizar o PSP à sua própria prática.

Porém, é preciso atentar para o fato que uma ferramenta mal desenvolvida seria uma inconveniência a seus usuários. [HUM 95] Uma ferramenta de apoio do PSP deve ser transparente com respeito à manipulação de tarefas, tal que os engenheiros possam se focalizar nos seus produtos em vez de nos seus processos.

6.3.1 Ferramentas disponíveis

Psptool 0.6 é um *script* TCL-TK de plataforma independente escrito por Andrew Worsley. [WOR 97] Provê uma implementação do PSP 2.1. A ferramenta caracteriza um painel de controle principal que administra o resumo de plano de projeto, cria exemplos de bugs e monitora o tempo.

Timelog é uma implementação baseada em Java do Log de Registro de Tempo do PSP, escrito por Christoph Clemens como parte de uma tese de Mestrado. Implementa todos os componentes padrões do Log de Registro de Tempo do PSP. Além disso, calcula a métrica "Tempo Em Fase", requerida pelo PSP.

Timmie é uma ferramenta de monitoramento de tempo disponível pelo Grupo de IPD na Universidade de Karlsruhe. É um programa que monitora o tempo gasto em diferentes projetos.

O PSPToolkit é um conjunto de ferramentas que pretende facilitar o gerenciamento do PSP. Ele inclui opções para monitoração dos tempos, anotação de defeitos, planejamento, visualização de resumos e gráficos de dados de projeto, manutenção de um histórico de projetos antigos e contagem de LOCs.

O **PSP Studio** foi desenvolvido em 1997 como um projeto do Departamento de Informática na East Tennessee State University. Roda na plataforma Windows e é conduzido usando o sistema de gerenciamento de banco de dados SQL Anywhere. Este produto apóia mais tarefas do PSP do que outras ferramentas. É projetado especificamente para usar com o curso de PSP. O atributo mais notável de PSP Studio é como pode ser adaptado para executar o PSP a quaisquer níveis, desde o nível 0 até o nível 3.

6.4 Pesquisa atual

O PSP tem dois projetos de pesquisa principais em desenvolvimento: [COU 98]

Team *Software* Process (TSP)

Watts Humphrey está trabalhando em um processo chamado Processo de *Software* de Time (TSP). O TSP aplica princípios de CMM e métodos do PSP a um projeto de *software* baseado em times (grupos). Ele é projetado para permitir a um grupo de PSP treinar os passos de administração de metas, definição de papéis, planejamento e administração de riscos.

One Person Project

O objetivo do processo do *One Person Project* é criar um processo que proveria cobertura para as áreas de prática fundamentais não suportadas pelo PSP. O projeto é uma tese que é desenvolvida como um projeto em comum entre a University of Calgary e os Sistemas de Informação Avançados (*Advanced Information Systems*) em Peoria, Illinois.

O OPP contém processos de *software* adicionais que são projetados para apoiar as seguintes KPAs: [FRA 97]

- Administração de Configuração de *software*;
- Garantia de Qualidade de *software*;
- Administração de Requerimentos;
- Definição de Processo da Organização;
- Foco de Processo de Organização.

É pretendido que o OPP seja usado por um engenheiro que está trabalhando em um projeto individual que não tem o apoio de uma organização madura e que não tem o controle sobre o seu ambiente de trabalho.

7 A opinião de *experts*

Com o propósito de sanar algumas dúvidas referentes ao PSP, o autor do presente trabalho consultou, por e-mail[21], dois dos maiores especialistas de PSP no Brasil: Átila Belloquim e Jones Oliveira de Albuquerque.

Átila Belloquim é especialista em Metodologias de Desenvolvimento de Sistemas, Mestrando em Administração de Empresas pela USP, Conferencista em eventos de informática e Professor de PSP no curso de pós-graduação em Engenharia de *Software* do SENAC-SP. Atualmente é Gerente de Serviços de Treinamento da Choose Technologies e Membro da Coordenação do SPIN Brasil (Grupo de Usuários SEI/CMM) e do Conselho Editorial de DevMag. (atila@choose.com.br).

Jones Oliveira de Albuquerque é graduado e Mestre pelo Departamento de Informática da Universidade Federal de Pernambuco (UFPE). Atualmente é Professor Assistente I na Universidade Federal de Lavras (UFLA) e doutorando no Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (UFMG). (joa@comp.ufla.br)

A seguir, são transcritas as perguntas feitas aos professores acima citados, com as respectivas respostas:

Pergunta 1. Em que situação se encontra o PSP no Brasil? Em quais Universidades ou Instituições estão sendo feitos estudos sobre essa metodologia? Quais as empresas que já utilizaram ou estão utilizando o PSP no Brasil?

Prof. Jones - No Brasil o pessoal do CITS Curitiba estava treinando pessoal nisso. Quanto as instituições, vejo três: UFMG, UFPE e UFLA. Quanto as empresas, não tenho muitos dados: uma empresa em Belo Horizonte que eu implantei, uma no recife iniciou estudos mas não sei se concluiu a implantação do modelo, existem varias usando CMM, mas PSP não...

Prof. Átila - O PSP está em seu início no Brasil. Há algumas pessoas estudando o assunto academicamente, mas as empresas por enquanto estão apenas procurando se inteirar do assunto. Dou um curso de PSP na pós-graduação do SENAC em Qualidade de *Software*, e também temos o curso aqui na Choose. Alguns de nossos clientes estão interessados, e pretendemos estar ajudando alguns deles a implantar o PSP em breve.

2. Encontrei pouquíssimo material sobre o assunto no Brasil. Na sua opinião, porque o PSP é tão desconhecido?

[21] Correspondência por *e-mail* com o Professor Jones Oliveira de Albuquerque (joa@comp.ufla.br) e Professor Átila Belloquim (atila@choose.com.br).

Prof. Jones - No Brasil, até dois anos atrás eu era a única pessoa que escrevia sobre o assuntou: http://www.comp.ufla.br/~joa/opcoes_menu/colegio/publicacoes/publicacoes.
html. Um motivo seria que as empresas de SW ainda não despertaram para a organização e eficiência em previsão de tarefas! As empresas de SW raramente cumprem exatamente o que planejam (dados da fiesp/sebrae) a falta de disciplina de nos brasileiros é um fator também que torna o uso de PSP uma tarefa árdua....

Prof. Átila - O PSP é desconhecido porque é muito novo, e nossa comunidade acadêmica é às vezes um pouco lenta...

3. Mundialmente, como anda o estudo do PSP? As empresas têm adotado o PSP? Não lhe parece que, passados cinco anos desde o lançamento do livro de Humphrey, seu método não "pegou"?

Prof. Jones - Você tem lido a *IEEE-Software*? As empresas tem usado e se preocupam com isso, vide http://www.computer.org/*software*/. Esta e próximas edições serão dedicadas a processo de *software* e uma edição será exatamente em PSP! se não estivesse em uso não teria fórum para discussão...

Prof. Átila - Nos Estados Unidos, muitos dos clientes tradicionais do CMM estão usando o PSP. Assim como o CMM, a implantação do PSP exige um grande compromisso das empresas, que às vezes elas não estão dispostas a assumir. CMM e PSP dizem respeito a mudança cultural, que é um processo MUITO lento, de modo que 5 anos não são nada neste contexto.

4. Na sua opinião, o que está faltando para que o PSP tenha mais aceitação? O senhor não acha que o método de Humphrey exagera no número de formulários, *scripts*, modelos e controles?

Prof. Jones - Sim concordo com você, uma maneira seria tentar capturar os dados de maneira automática... Mas em conversa com o próprio Humphrey constatei que ele não gosta da idéia da automatização, mas no meu ponto de vista os relatórios é que desmotivam os programadores...

Prof. Átila - Para maior aceitação precisaremos de maior divulgação tanto do método quanto de seus benefícios. Os formulários dão mesmo trabalho, e devem ser automatizados (há ferramentas para isso), para reduzir a "papelada"; mas não há como se atingir um processo disciplinado sem a coleta dos dados que o PSP indica.

5. Qual sua opinião sobre o PSP? Na prática ele realmente funciona? O senhor usa esse método?

Prof. Jones - Na prática, como esta definido hoje, não funciona... Sim uso, mas é uma versão reduzida dele... Temos trabalhado aqui na UFLA com uma tentativa de torná-lo prático... Mas isso demorará um pouco...

Prof. Átila - O método é muito bom e eu o utilizo numa versão customizada para o tipo de trabalho que eu faço, que não é programação. O importante é a pessoa aprender o método, modificá-lo e, possivelmente, simplificá-lo para o tipo de atividade que realiza. O PSP é ensinado "completo", o que não quer dizer que todo mundo deva adotar todas as suas sugestões ao mesmo tempo!

8 Comentários finais

> "Uma pessoa inteligente resolve um problema.
> Uma pessoa sábia o evita".
> Einstein

O livro de Humphrey e seus exercícios de programação ensinam muito sobre engenharia de *software*. O livro não se limita aos assuntos de processo pessoal. Nele, Humphrey discute temas importantes da Engenharia de *Software* e da Ciência da Computação, como um todo. Concentrando-se no PSP, ele mostra como lidar quantitativamente com o processo de desenvolvimento de *software*. Isto rende uma riqueza de dados e ferramentas que o desenvolvedor pode usar para administrar o seu trabalho e melhorar as suas tarefas de programação. O livro também expõe uma larga variedade de práticas que podem ser usadas para analisar e melhorar o trabalho do engenheiro de *software*.

Por outro lado, muito esforço é exigido para seguir esse método. Não apenas ao executar o trabalho e ao estudar os próprios métodos, mas também na interferência com as práticas atuais. Se essas práticas foram estabelecidas após muitos anos de trabalho, uma mudança para métodos novos e pouco conhecidos pode causar um alto grau de confusão e frustração ao aprende-los.

Um grande cuidado deve ser tomado com relação aos perigos de interpretar errado a necessidade por processos definidos como uma desculpa para a burocracia. Embora Cook [COOK *apud* [CAS 99]] apóie fortemente abordagens como o ISO9000 e o Modelo de Maturidade de Capacidade, ele adverte:

> *"Lembre-se que o software de qualidade é o fim, e que o processo é o meio. Se mantivermos nossos olhos no objetivo e modificamos o processo para nos permitir alcançá-lo, então software de qualidade pode ser produzido".*

A questão da produtividade também é importante. Provavelmente, os benefícios do PSP seriam tanto mais claros se as LOC/hora fossem mantidas ou melhoradas enquanto que os defeitos nos testes fossem reduzidos. Essa questão trata também do grande volume de controles e formulários, bem como da questão da automatização completa do PSP. Talvez uma melhor análise de custo/benefício seja importante.

Existem evidências experimentais de seus benefícios, mas para convencer, uma avaliação mais cuidadosa teria que avaliar o efeito de introduzir o PSP nos objetivos empresariais de uma organização, não simplesmente no tocante à qualidade de *software*. [CAS 99]

Apesar de tudo, um grande conhecimento e perspicácia sobre os métodos pessoais de programação são conseguidos. Como Humphrey diz, a diferença ao programar agora é como dirigir à noite com as luzes do carro acesas. Pode-se ver quando melhorar e por quanto. Não há nenhum método instantâneo para a melhoria. Mas com estes métodos e muito trabalho, existe uma maneira de se fazer progresso continuamente. [HUM 95]

Como, ironicamente, diz [CAS 99], os desenvolvedores podem não gostar da disciplina exigida pelo "Processo de *Software* Puritano" (*Puritanical Software Process*), mas a atitude de cavalheiro alternativa para o desenvolvimento é sentenciada ao fracasso. Não é uma questão de criatividade contra disciplina, mas de trazer disciplina ao trabalho de forma que a criatividade possa acontecer.

Outras considerações:

Com relação ao aprendizado do PSP, não houve grandes dificuldades para sua compreensão no que se refere aos conceitos de Engenharia de *Software*, informática em geral, programação e estatística. Isto, provavelmente demonstra que o curso de informática da UFPEL fornece uma boa bagagem de conhecimentos. É importante, porém, ressaltar, que não efetuou-se um estudo completo e profundo do livro-texto, já que não era esse o objetivo do presente trabalho. Foram feitos os exercícios propostos até o nível PSP1.1 – o chamado PSP Básico, para ter uma compreensão prática e básica do problema.

Outra conclusão facilmente obtida é que é inviável utilizar o PSP, no trabalho ou nos estudos, sem antes aprendê-lo muito bem.

Para afirmar-se efetivamente junto à comunidade acadêmica e profissional, o PSP precisa resolver algumas questões de ordem prática, como o problema de controle dos dados, que é feito de forma não ordenada, isto é: registra-se o tempo no Log de tempo, depois volta-se para o Sumário de Plano, depois preenche-se o Log de defeito, depois produz-se código, etc. Possivelmente uma boa ferramenta para tratar de forma automatizada os dados possa resolver esses problemas. Também, a questão da revisão do código em busca de erros sintáticos (que o compilador pode capturar), é passível de críticas. Com compiladores tão eficientes disponíveis hoje em dia o autor acredita que é mais razoável procurar erros lógicos.

Apesar do CMM ser conhecido no Brasil, o mesmo não pode ser dito com relação ao PSP. Para saber o porquê dessa situação, dois pesquisadores da área no Brasil[22] foram contatados: o Professor Jones Albuquerque, da Universidade Federal de Lavras, acredita que um motivo seria que as empresas de *software* ainda não despertaram para a organização e eficiência em previsão de tarefas. Segundo o professor "as empresas de *software* raramente cumprem exatamente o que planejam (dados da FIESP/SEBRAE) e a falta de disciplina de nós, brasileiros, é um fator também que torna

[22] Vide Capítulo 9.

o uso de PSP uma tarefa árdua". Como pode ser notado, é possível que o problema não seja o método em si. Já o Professor Átila Belloquim, do SENAC - SP, acredita que um dos problemas é a lentidão da comunidade acadêmica para absorver novas idéias. Em termos globais, o Professor Átila ressalta que o CMM e o PSP dizem respeito a mudança cultural, que é um processo muito lento, de modo que cinco anos não são nada nesse contexto.

Embora no Brasil a comunidade de *software* não tenha ainda despertado para a importância do PSP, o mesmo não pode ser dito da comunidade internacional, como pode ser comprovado no presente trabalho. Além disso, uma das matérias de capa da *IEEE Software*[23], na sua edição de **novembro/dezembro de 2000**, será: "*The Personal Software Process*", por Watts Humphrey.

[23] Revista eletrônica da *IEEE Computer Society*

9 Literatura comentada

Em *Data Quality Problems in the Personal Software Process*, tese de mestrado na University of Hawaii, de 1998, Anne Disney relata que a sua experiência pessoal no uso do PSP em atividades industriais e acadêmicas lhe causou o desejo de saber sobre a qualidade de duas áreas do PSP: coleção e análise. Para investigar estas questões ela construiu uma ferramenta para automatizar o PSP e então examinou 89 projetos completados por nove pessoas usando o PSP em uma conotação educacional. (http://csdl.ics.hawaii.edu/)

Em *Evaluation of PSP in the Undergraduate Education*, Stefan Olofsson examinou o uso do PSP em um curso universitário na Umea University. A conclusão mais importante a que ele chegou foi que aprender os métodos do PSP toma muito tempo e não pode ser paralelizado com um curso de tempo integral. Esse trabalho também estudou o P-CMM (*People Capability Maturity Model*). [OLO 99a]

Em *Adjusting the Instruction of the Personal Software Process to Improve Student Participation*, Laurie A. Williams esboça um processo para administrar e produzir *software* de alta qualidade, reduzindo o rigor matemático e estatístico do PSP original, mas mantendo uma base sólida de práticas disciplinadas e uma filosofia de qualidade. O *paper* descreve como o Processo de *Software* Pessoal foi incorporado nos cursos de informática na Universidade de Utah. [WIL 97]

Em *Integrating the Personal Software Process (PSP) across the Undergraduate Curriculum*, Massood Towhidnejad e Thomas Hilburn sumarizam as atividades de incorporação do PSP no currículo de informática da Embry-Riddle Aeronautical University. O *paper* inclui uma descrição dos objetivos do projeto, uma discussão das atividades comprometidas com os estudantes, uma explicação de como as atividades foram integradas no currículo, uma descrição dos dados colecionados e uma avaliação do impacto e valor do projeto. [HIL 97]

Um resumo do PSP, bem como uma estratégia de implantação do PSP na Universidade, é feito por Dale Couprie em *The Personal Software Process – A Good Start for New Software Engineer*. O artigo, além de explicar e analisar de forma clara e concisa o PSP, apresenta uma sugestão de abordagem do PSP nos primeiros anos dos cursos de graduação universitária. [COU 98a]

Um dos mais interessantes textos encontrados pelo autor a respeito do PSP é o *paper* escrito por Chris Casey [CAS 99], da University of Central Lancashire, intitulado *Do Software Developers Need The Personal Software Process?* Nele, além de um breve resumo do PSP, Casey faz uma análise bastante crítica do método de Humphrey, apontando diversas problemas de forma bastante fundamentada.

Para outras referências, consultar a bibliografia.

10 Bibliografia

[BAG 97] BAGERT, Donald J. **A Comprehensive Integrated Three-Semester *Software* Engineering Sequence**. 1997. Disponível por WWW em http://fairway.ecn.purdue.edu/~fie/fie97/papers/1148.pdf (01/09/2000).

[BEL 00] BELLOQUIM, Átila, **PSP - O Processo de *Software* Pessoal**. 1999. Disponível por WWW em http://www.choose.com.br/artigos/html/textos/dm1098.htm (10/09/2000).

[BOH 81] BOHEM, Barry W. ***Software* Engineering Economics**. Englewood Cliffs, NJ: Prentice-Hall. 1981.

[CAS 99] CASEY, Chris. **Do *Software* Developers Need The Personal *Software* Process?** Disponível por WWW em http://www.uclan.ac.uk/facs/destech/compute/staff/casey/psp.zip (22/10/2000).

[CLE 97] CLEMENS, Christoph. **Timelog**. 1997. Disponível por WWW em http://mats.gmd.de:8080/clemens/java/timelog/index.html (20/09/2000).

[COR 00] CORDEIRO, Marco Aurélio. **Foco no Processo**. Companhia de Informática do Paraná - CELEPAR - Byte Número 100 - maio/2000. Disponível por WWW em http://www.celepar.br/batebyte/bb100/foco.htm (01/10/2000).

[COR 00a] CORDEIRO, Marco Aurélio, **O Panorama Atual da Indústria de *Software***. Companhia de Informática do Paraná - CELEPAR - Byte Número 97 – Agosto/2000. Disponível por WWW http://www.celepar.br/batebyte/bb97/panorama.htm página (01/10/2000.).

[CRI 00] CRISTIANO, Aroldo et al. **Processo Pessoal de *Software***. Disponível por WWW em http://campus.fortunecity.com/princeton/117/psp/psp.htm (20/09/2000).

[COU 98] COUPRIE, Dale. **Automated Support for a Custom Personal *Software* Development Process.** 1998. Disponível por WWW em http://sern.ucalgary.ca/students/theses/DaleCouprie/TableofContents.htm (10/09/2000).

[COU 98a] COUPRIE, Dale. **The Personal *Software* Process. A Good Start for the New *Software* Engineer**. Disponível por WWW em http://sern.ucalgary.ca/courses/seng/693/W98/couprie/major.html (21/10/2000).

[FRA 97] FRANKOVICH, J. F. **The One Person Project *Software* Process**. Masters Thesis - University of Calgary. Disponível por WWW em http://sern.ucalgary.ca/courses/seng/621/W97/johnf/thesis/thesis.htm, (22/07/2000).

[GLA 98] GLASS, R. L. ***Software* runaways, lessons learned from massive *software* project failures**. USA: Prentice Hall, 1998.

[GUI 99] GUIDO, Cardino. **ASPIDE – ESSI Project 24331 – Final Report Version 2.0,** *paper*. 1999. Disponível por WWW em http://www.esi.es/ (01/11/2000)

[HAY 97] HAYES, Will e Over, James W, **The Personal *Software* Process: An Empirical Study of the Impact of PSP on Individual Engineers**. 1997. Disponível por WWW em http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97tr001.pdf (01/09/2000).

[HEN 97] HENRY, J. **Personal *Software* Process Studio**. Department of Computer Science, East Tennessee State University. Disponível por WWW em http://www-cs.etsu-tn.edu/softeng/psp/index.htm (22/07/2000).

[HIL 97] HILBURN, T. e Towhidnejad, M., **Integrating the Personal *Software* Process across the Undergraduate Curriculum**. *Paper*.1997**.** Disponível por WWW em http://fairway.ecn.purdue.edu/~fie/fie97/papers/1148.pdf

[HUM 95] HUMPHREY, Watts S. **A Discipline for *Software* Engineering**. Reading: Addison Wesley Longman, Inc. 1995. 790p.

[JAL 99] JALOTE, Pankaj. **CMM in practice: processes for executing *software* projects at Infosys**. USA: Addison Wesley, 1999.

[JUN 00] JÚNIOR, José Barreto. **Qualidade de *Software***. Disponível por WWW em http://www.barreto.com.br/qualidade (20/09/2000).

[KOC 00] KOCH, Alan S. **PSP Industrial Use Data**. Disponível por WWW em http://www.askprocess.com/PSPData/index.html (01/10/2000).

[KOP 99] KOPANAS, V. et al. **Project PIBOP EP23825**. ESSI Process Improvement Experiment - PIBOP. *Paper*. 1999. Disponível por

WWW em http://www.esi.es/ (01/11/2000)

[LEI 00] LEITE, Jair C. **Notas de Aula de Engenharia de *Software***. Disponível por WWW em http://www.dimap.ufrn.br/~jair/ES/home.html (15/10/2000).

[MCT 00] MCT - Ministério da Ciência e Tecnologia. **Qualidade e Produtividade no Setor de *Software* Brasileiro**. Disponível por WWW em http://www.mct.gov.br/temas/info/dsi/qualidad/qualidad.htm (10/09/2000).

[MEN 98] MENEGOS, Stavros. **ESSI Project Number: 24158**. *Paper*. 1998. Disponível por WWW em http://www.esi.es/ (01/11/2000)

[OLO 99] OLOFSSON, Stefan. **PSPToolkit User Manual - Version 1.1**. Disponível por WWW em http://www.cs.umu.se/~olofsson/xjob/PSPtkManual.html (20/07/2000).

[OLO 99a] OLOFSSON, Stefan. **Evaluation of PSP in the Undergraduate Education**. 1999. Disponível por WWW em http://www.cs.umu.se/local/exjobb/jubo4.html (20/07/2000).

[PAU 93] PAULK M.C., Bill Curtis e M.B. Chrisis. **Capability Maturity Model for *Software*, Version 1.1**. *Software* Engineering Institute Technical Report, CMU/SEI-93-TR 1993. Disponível por WWW em http://www.sei.cmu.edu (15/10/2000).

[PUT 97] PUTNAM, L. H. e MYERSM, W. **Industrial strength *software* – effective management using measurement**. USA: IEEE Computer Society Press, 1997.

[RIC 00] RICO, David F. **The personal *Software* Process In The Internet Age.** Disponível por WWW em http://davidfrico.com/psppaperdoc.htm (01/09/2000).

[ROD 00] RODRIGUES, Alex S. D. **PSP – Personal *Software* Process**. Disponível por WWW em http://www.geocities.com/Hollywood/Boulevard/1429/psp.htm (20/09/2000).

[SAN 97] SÁNCHEZ, Valentim et al. **TRAPSP: Advanced Multimedia and Distance Learning for SME Teams using the PSP**. Disponível por WWW em http://www.esi.es/Publications/Articles/Art/97ssc01.html, (01/10/2000).

[SEI 00] SEI - *Software* Engineering Institute. **What is PSP?** Disponível por WWW em http://www.sei.emu.edu/psp/WhatPSP.htm (01/09/2000).

[SEI 00a] SEI - *Software* Engineering Institute. **Personal *Software* Process^{SM} (PSP^{SM}) and Team *Software* Process^{SM} (TSP^{SM}) Results**. Disponível por WWW em www.sei.cmu.edu/tsp/results.html (01/10/2000).

[STA 95] STANDISH GROUP. **Chaos**. 1995. Disponível por WWW em www.standishgroup.com/chaos.html (04/10/2000).

[UMD 97] University of Massachusetts Dartmouth. **Personal *Software* Process**. Disponível por WWW em http://www2.umassd.edu/swpi/Personal*Software*Process/psp.html (20/09/2000).

[WIL 97] WILLIAMS, L. A. **Adjusting the Instruction of the Personal *Software* Process to Improve Student Participation.** 1997. http://fairway.ecn.purdue.edu/~fie/fie97/papers/1305.pdf (25/09/2000).

[WOR 97] WORSLEY, A. **Psptool 0.6**. Disponível por WWW em http://www.neosoft.com/tcl/ftparchive/sorted/math/psptool0.5p1 (05/09/2000).

[WOR 96] WORSLEY, A. **What are the benefits of the PSP *Software* Process?** 1996. Disponível por WWW em http://www3.cm.deakin.edu.au/~peter/PSP_data/talk.html acessada em (05/09/2000).