UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

PCU|$_{PSP}$: Uma Estratégia que Promove Agilidade na Gerência de Projetos de Software

ARNALDO NAPOLITANO SANCHEZ

ORIENTADORA: PROF^{A}. DRA. SANDRA CAMARGO P. F. FABBRI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciências da Computação, área de concentração: Engenharia de Software

São Carlos/SP
Janeiro/2008

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

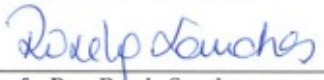
"PCU \|PSP: Uma estratégia que promove agilidade na gerência de projetos de software"

ARNALDO NAPOLITANO SANCHEZ

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa. Dra. Sandra Camargo P. Ferraz Fabbri
(Orientadora – DC/UFSCar)



Profa. Dra. Rosely Sanchez
(ICMC/USP)

Prof. Dr. Fábio Kon (IME/USP)

São Carlos
Fevereiro/2008

Ao meu grande amor Ana Elisa, por ser uma pessoa maravilhosa que escolheu estar ao meu lado.

Agradecimentos,

Essa dissertação de mestrado não seria possível sem o apoio de pessoas maravilhosas que estiveram ao meu lado.

Em primeiro lugar, eu agradeço à minha orientadora, Profa. Dra. Sandra Fabbri. Por ter me apoiado em todos os momentos, pela sua paciência, e pela sua forma carinhosa de orientar.

À Profa. Dra. Rosely Sanchez que me ajudou não somente como professora e pesquisadora, mas também, como uma amiga.

Aos sócios e funcionários da Linkway e da NBS, pela compreensão, dedicação e seriedade na execução dos trabalhos aqui relatados. Sem os quais essa dissertação não seria possível.

Aos meus amigos do laboratório que me ajudaram nos momentos mais difíceis: Denis, Renan e Luiz.

A minha amada mãe, Maria José, exemplo de mulher. Com seu carinho, sempre atenta a me apoiar. Ao meu pai por ter me dado às bases espirituais que me sustentam.

Aos meus amigos Lígia, Rodolfo, Alice Cristina Corradini, Jefferson Curradini, Vesta de Antares, Apolo de Antares e Guilherme pelo apoio irrestrito.

Ao meu grande amor, Ana Elisa, por estar ao meu lado em todos os momentos, me incentivando e me fazendo lutar pelos meus sonhos.

E a Deus, pela oportunidade e por me cercar de pessoas maravilhosas que permitiram que esse trabalho fosse concluído.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

Tabela 1.1 - Lista de Abreviaturas

APF	Análise de Pontos por Função
PCU	Pontos por Caso de Uso
CMM	*Capability Maturity Model*
IEEE	*Institute of Electrical and Electronics Engineering*
PSP	*Personal Software Process*
UML	*Unified Modeling Language*
API	*Application Program Interface*
SW-CMM	*Capability Maturity Model for Software v2.0 draft C*
SECM	*Systems Engineering Capability Model*
IPD-CMM	*Integrated Product Development Capability Maturity Model v 0.98*
EIA	*Electronic Industries Alliance*
CMMI	*Capability Maturity Model Integration*
MPS.BR	Melhoria de processo de Software Brasileiro
IDE	*Integrated Development Environment*
ALI	Arquivo Lógico Interno
AIE	Arquivo de Interface Externa
DER	Dado Elementar Referenciado
RLR	Registro Lógico Referenciado
EE	Entrada Externa
SE	Saída Externa
CE	Consulta Externa
PFN	Pontos de Função Não Ajustados
FAV	Fator de Ajuste do Valor
MPE	Micro ou Pequena Empresa.
SOFTEX	Associação para Promoção da Excelência do Software Brasileiro
FINEP	Financiadora de Estudos e Projetos
MA-MPS	Método de Avaliação para Melhoria de Processo de Software
MN-MPS	Modelo de Negócio para Melhoria de Processo de Software
AP	Atributos de Processo
NESMA	*Netherlands Software Metrics Association*
BID	Banco Interamericano de Desenvolvimento
MCT	Ministério da Ciência e Tecnologia
JPA	*Java Persistence API*
JSF	*Java Server Faces*

Resumo

Cenário: buscando implantar modelos de qualidade de processo de desenvolvimento de software em pequenas empresas, vários trabalhos foram desenvolvidos em parceria com os centros de pesquisa da Universidade Federal de São Carlos e da USP São Carlos. Nesta busca identificou-se a necessidade de melhorar o processo de gerenciamento de projetos. ***Objetivo:*** este trabalho apresenta uma estratégia que facilita o gerenciamento de projetos de software por meio do ajuste do método de Pontos por Casos de Uso (PCU), para que este reflita efetivamente as características da equipe de desenvolvimento, de forma que o planejamento do sistema a ser desenvolvido seja mais próximo da realidade. Esse ajuste provém da aplicação de atividades do processo PSP (*Personal Software Process*) e de uma constante avaliação das atividades de desenvolvimento, em conjunto com a atualização e o controle do plano de projeto elaborado. ***Método:*** essa estratégia foi extraída da aplicação sistemática do método PCU e do processo PSP em conjunto, no ambiente de pequenas empresas, com a finalidade de acompanhar o progresso do projeto. Para tanto, identificou-se a necessidade de ajustar, ao longo do projeto, o Nível De Esforço (NDE) por Ponto de Caso de Uso, e também, de ajustar os Fatores Ambientais (previstos no método PCU) para cada desenvolvedor. O monitoramento do projeto através do ajuste das variáveis do método PCU baseado nas atividades do PSP, resultou na elaboração da Estratégia $PCU|_{PSP}$ aqui proposta. ***Resultados:*** a estratégia está sendo aplicada na prática em pequenas empresas de software, como mostram dois estudos de caso apresentados no trabalho e os resultados obtidos dão indícios de sua contribuição para a melhoria da qualidade do processo de gerenciamento de projetos de software, principalmente no que diz respeito às atividades de planejamento e controle. ***Conclusão:*** pelos resultados até agora obtidos, a estratégia atingiu o objetivo de facilitar a implantação de modelos de qualidade de processo, como o MPS.BR, no que se refere ao gerenciamento de projetos. Além disso, ela se ajusta perfeitamente ao método ágil *Scrum*, deixando a gerência de projeto bastante ágil e controlável, o que é um ponto positivo, principalmente para empresas de pequeno porte.

Abstract

***Background*:** aiming to implement software processes quality models for small companies, several projects were developed in collaboration with Universidade Federal de São Carlos and USP São Carlos research centers. In this context, it was identified the need of improving the process of project management. ***Aim*:** this work presents a strategy that facilitates the management of software projects adjusting the Use Case Points method so that it effectively reflects the development team characteristics and the system planning to be developed be close the reality. This adjust comes from the application of the PSP (Personal Software Process) process activities and from a constant evaluation of the activities in course, together the actualization and control of the elaborated project planning. ***Method*:** this strategy was extracted from the systematic application of both the PCU method and PSP process, in the small companies' environment, so that the project progress can be always evaluated. On account of this, it was identified the necessity of adjust the Effort Level (NDE – in Portuguese) per Use Case Point as well as the environmental factors of the PCU method per developer. The monitoring of the project through the PCU method variables adjustment, based on the PSP activities, resulted in the definition of the $PCU|_{PSP}$ Strategy, proposed here. ***Results*:** the strategy is being actually applied in small software companies, as shown in the two case studies presented in this work. The obtained results indicate its contribution to the quality improvement of the management software projects, mainly in relation to the planning and control activities. ***Conclusion*:** based on the obtained results, the strategy has achieved the objective of facilitating the implementation of process quality models, like the MPS.BR, only in respect to project management. Besides, the strategy is perfectly adjusted to the *Scrum* agile method, making agile and controllable the project management, what is a positive point, mainly in the context of small companies.

Capítulo 1 – Introdução

1.1 - Contexto e Motivação

Segundo Pressman [PRESSMAN,2006], as estimativas de prazos e custos, freqüentemente, são imprecisas. A produtividade dos desenvolvedores de software não tem acompanhado a demanda, e a qualidade de software muitas vezes é inferior à desejada, deixando os usuários finais do produto, em muitos casos, insatisfeitos.

Com o objetivo de melhorar a qualidade do processo de desenvolvimento de software e, conseqüentemente, a qualidade do produto final, surgiram os modelos de qualidade de processo como o CMM [PAULK,1993], CMMI [CMMI,2006] e o MPS.BR [MPSBR,2007]. No entanto, eles especificam “o que” deve ser feito para que se tenha qualidade, mas não “como” deve ser feito. Especificar modos de fazer limita o uso dos modelos de qualidade nas mais diversas organizações. Questões relacionadas ao porte da organização, cultura organizacional, recursos disponíveis e experiência da equipe impõem características particulares ao processo de desenvolvimento [ROCHA,2001]. No entanto, sem uma guia clara de desenvolvimento que possa ser entendida por um desenvolvedor, o trabalho de desenvolvimento de software se torna muito difícil [MARINESCU,2004]

No caso das pequenas empresas brasileiras, principalmente, a baixa capacidade de investimento não permite a implantação de modelos de qualidade e certificações, pois estas possuem alto custo. O custo, em média, no Brasil é de R$ 250 mil por nível de maturidade do CMMI [CTXML,2006], que é o modelo de qualidade mais utilizado internacionalmente. Dessa forma, a exigência desse investimento inviabiliza que as pequenas empresas participem desse mercado.

Para contornar um pouco esse problema no contexto nacional, foi criado o modelo de qualidade brasileiro, MPS.BR [MPSBR,2007], que se adequou às condições do mercado nacional, propondo dar atenção especial às micro e pequenas empresas, a fim de possibilitar que estas possam investir em suas certificações. Além disso, esse modelo propõe ser compatível com os padrões de qualidade aceitos internacionalmente e permitir a implementação das disciplinas de engenharia de software, conforme o contexto das empresas brasileiras.

No entanto, ainda permanece o "como fazer", o que torna motivo de pesquisa e interesse buscar meios e alternativas que possam facilitar a implantação dos modelos de qualidade, associando-os a estratégias que determinem disciplinas de engenharia de software.

1.2 – Objetivo

O foco desta pesquisa está no gerenciamento de projetos, que é uma das disciplinas de engenharia de software e se encontra como um dos itens obrigatórios de todos os modelos de qualidade.

Com o objetivo de contribuir com o "como fazer" na questão de gerência de projetos para implantação dos modelos de qualidade, define-se aqui a estratégia $PCU|_{PSP}$, que determina um processo de aplicação das técnicas Pontos por Caso de Uso (PCU) e *Personal Software Process* (PSP) [HUMPHREY,1995], de forma a facilitar essa gerência, principalmente no contexto de pequenas empresas. Com a integração dessas duas técnicas, a estratégia $PCU|_{PSP}$ determina uma série de etapas que permite abordar o sistema a ser desenvolvido por partes, distribuir as tarefas entre os desenvolvedores, fazendo um planejamento do software de acordo com as características de cada um, e fazer com o que o próprio desenvolvedor consiga medir seu trabalho e sua produtividade, de forma a permitir um acompanhamento do plano de

maneira bem sistemática, o que permite a realização de ações corretivas em tempo. Além disso, *gerente* de projeto e desenvolvedores precisam manter uma grande interação, o que permite também que o *gerente* atue como um treinador do desenvolvedor.

Assim, a estratégia $PCU|_{PSP}$ pode facilitar o cálculo do custo do sistema a ser desenvolvido e auxiliar o desenvolvimento profissional dos desenvolvedores.

1.3 - Histórico do Trabalho

A empresa Linkway, caracterizada na seção 6.2, busca melhorar seus processos de software desde o ano 2000, quando a preocupação com a qualidade se tornou uma meta para os *gerentes* da empresa. A busca da qualidade pela Linkway, e a determinação dos pesquisadores em levar o fruto do trabalho dos centros de pesquisa para as empresas, contribuiu para a união da equipe da Linkway, com os pesquisadores do grupo de Engenharia de Software do Departamento de Computação da UFSCar (Universidade Federal de São Carlos), e com os pesquisadores do ICMC (Instituto de Ciências Matemáticas e de Computação) da USP São Carlos. Esta união possibilitou a realização de algumas dissertações de mestrado em parceria com a Linkway. Os trabalhos realizados neste contexto estão resumidos abaixo, em ordem cronológica.

- **2000: Marchetti** [MARCHETTI,2000] determinou um instrumento para caracterizar os pontos fortes e fracos de um processo de software de uma empresa de pequeno porte, como base no nível 2 do *Software* CMM.
- **2002: Tavares** [TAVARES,2002] identificou o processo de desenvolvimento de software da Linkway e, com base em uma ferramenta para planejar a melhoria do processo, proposta na dissertação da Marchetti [MARCHETTI,2000], foi estabelecida uma estratégia para implantação da qualidade no processo de desenvolvimento de software da Linkway

- **2002: Brogini** [BROGINI,2002], propôs uma Abordagem Evolucionista para a Garantia da Qualidade de Software, denominada GQS-AE. Ela consiste de uma estratégia a ser utilizada para manipulação, organização e gerenciamento de todos os processos (e seus produtos) praticados por uma empresa de desenvolvimento de software.
- **2004: Cândido** [CÂNDIDO,2004] apresenta uma simplificação da contagem detalhada de pontos por função promovida pelo IFPUG (*International Function Point Users Group*), utilizando como princípio as idéias de simplificação sugeridas pela NESMA (*Netherlands Software Metrics Association*).
- **2004: Silva** [SILVA,2004] contribuiu com a melhoria de processo de software, principalmente das Empresas de Pequeno Porte, explorando algumas abordagens que podem facilitar essa tarefa.

Através da contribuição das pesquisas realizadas, a Linkway elevou a qualidade de seus processos de desenvolvimento de software e criou uma consciência de qualidade e melhoria contínua dentro da equipe de desenvolvimento.

As pressões do mercado por resultados rápidos, somada a consciência de qualidade fomentada pela parceria com as universidades, levaram a Linkway e o grupo de Engenharia de Software da UFSCar a desenvolver uma estratégia ágil para melhorar a gerência de projetos de software.

Esta estratégia nasceu da necessidade de unir o planejamento com o controle do projeto de software. A meta foi desenvolver uma maneira de fazer estimativas freqüentes, para medir o progresso do trabalho ao longo de todo o projeto de software. Essa estimativa deveria também ser calibrada a cada entrega de uma parte do software.

Em paralelo a esse trabalho, a Linkway, em função dos projetos de

pesquisa anteriores, estava em processo de implantação do PSP (*Personal Software Process*) [HUMPHREY,1995], para melhorar a qualidade da equipe de desenvolvimento. Assim, a união do método PCU (Pontos por Casos de Uso) [KARNER,1993], que já vinha sendo utilizado na empresa, com o processo PSP se tornou uma conseqüência natural de trabalho, levando à identificação e definição da Estratégia PCU|$_{PSP}$ proposta neste trabalho.

Como o objetivo da Linkway é a implantação da qualidade e uma possível certificação, buscou-se um método ágil que desse suporte aos processos de gerência de projetos. Desta forma, foi possível perceber que a união da Estratégia PCU|$_{PSP}$ com o método ágil *Scrum* [SCHWABER,2004] se ajusta de forma bastante natural, facilitando a implantação do processo Gerência de Projetos do nível G do MPS.BR [MPSBR,2007].

Ressalta-se que a estratégia foi apresentada no SBQS 2007, VI Simpósio Brasileiro de Qualidade de Software, por meio do artigo “PCU|$_{PSP}$ – Uma Estratégia para ajustar Pontos por Caso de Uso por meio do PSP em empresas de pequeno porte”.

1.4 - Organização do Trabalho

Este trabalho está organizado em 7 capítulos como descrito a seguir: No Capítulo 2 apresenta-se a revisão bibliográfica dos modelos de qualidade de processo CMMI, do modelo de qualidade brasileiro MPS.BR, e do processo PSP, que é um modelo de qualidade de processo pessoal, utilizado neste trabalho. Apresenta-se também a ferramenta *Process Dashboard*, que dá suporte à implantação do PSP.

No Capítulo 3 apresenta-se a revisão bibliográfica sobre Planejamento de Software. Para tanto, é apresentada a métrica APF (Análise por Pontos de Função), o método PCU (Pontos por Caso de Uso), alguns conceitos sobre a gerência de riscos e os itens essenciais para montar um plano de projeto de software.

No Capítulo 4 apresenta-se uma revisão sobre Métodos Ágeis, abordando-se os princípios ágeis, com ênfase no planejamento e gerenciamento ágil. Também são apresentadas as principais características do método *Scrum*.

No Capítulo 5 descreve-se a Estratégia PCU|$_{PSP}$ que é a proposta deste trabalho. Além disso, explora-se o uso do PSP para ajudar na gerência de projetos e faz-se uma relação da estratégia com o método *Scrum* e com o modelo de qualidade MPS.BR.

No Capítulo 6 são relatados três estudos de caso em duas pequenas empresas de software. Nos dois primeiros, a ênfase é dada ao controle do projeto de software, através dos constantes ajustes do NDE (nível de esforço), que retrata o desempenho do desenvolvedor ao longo do projeto. No terceiro estudo de caso, a ênfase é dada no gerenciamento dos desenvolvedores, por meio do PSP.

Por fim, no Capítulo 7, apresentam-se a conclusão, as contribuições do trabalho, as lições aprendidas e os trabalhos futuros a serem executados nesta área.

Capítulo 2 – Modelos de Qualidade

2.1 Considerações iniciais

A proposta deste trabalho tem um impacto direto na qualidade do processo de software, uma vez que trata de atividades de planejamento, as quais têm um destaque especial em todos os modelos de qualidade de processo propostos. Nos diversos modelos essas atividades são representadas por processos específicos que devem ser satisfeitos para que uma empresa inicie um programa de melhoria e saia do estágio caótico.

Como foi mencionado no capítulo anterior, no caso das empresas consideradas no contexto deste trabalho, a busca pela melhoria do processo teve início há vários anos, com vistas, inicialmente, ao modelo CMM – *Capability Maturity Model* [PAULK,1993], passando depois ao modelo CMMI – *Capability Maturity Model Integration* [CMMI,2006] e atendo-se agora ao modelo MPS.BR – [MPSBR,2007], pelo fato deste ser um modelo com alvo nas pequenas e médias empresas nacionais.

Além desses modelos que dão suporte ao processo como um todo, as empresas aqui tratadas sempre investiram na melhoria pessoal de sua equipe de desenvolvimento e, neste caso, tem-se usado como suporte o modelo de qualidade de processo individual PSP – *Personal Software Process* [HUMPHREY,1995] que, além de dar subsídios ao aperfeiçoamento dos desenvolvedores das empresas, foi essencial para a definição da proposta apresentada neste trabalho.

Assim, na seção 2.2 serão apresentados os principais conceitos sobre o CMMI, pois além de ter sido alvo das empresas durante muito tempo, é a base do modelo MPS.BR, o qual é apresentado na seção 2.3. Na seção 2.4 apresenta-se o modelo de qualidade pessoal, PSP, que é parte fundamental da estratégia aqui proposta.

Nessa seção dá-se também uma visão geral da ferramenta Process Dashboard, que dá suporte ao uso desse processo e que é usada rotineiramente pelos desenvolvedores das empresas abordadas neste trabalho. Na seção 2.5 são apresentadas as considerações finais.

2.2 CMMI

O CMMI – *Capability Maturity Model Integration* [CMMI,2006], foi criado para suprir a necessidade de algumas empresas que não eram necessariamente da indústria de software, mas desenvolviam software e também produziam outros produtos como carros, equipamentos eletrônicos, etc. Nesses casos, essas organizações eram obrigadas a usar dois ou mais tipos de modelos de qualidade de processo. O propósito do CMMI é fornecer um modelo que permita a melhoria do produto e do processo e reduzir redundâncias e inconsistências resultantes da utilização de vários modelos de qualidade usados separadamente. O CMMI agrupa três modelos de qualidade em um único *framework*, incorporando também compatibilidade com a norma ISO/IEC 15504 [CHRISSIS,2003].

O CMMI possui quatro áreas de conhecimento disponíveis, a saber [CHRISSIS,2003]:

- **Engenharia de Sistemas** – essa disciplina engloba o desenvolvimento de sistemas, podendo ser ou não software. Os engenheiros de sistema focalizam na transformação das necessidades do consumidor, expectativas e restrições para o desenvolvimento dos produtos e suporte aos mesmos.
- **Engenharia de Software** – essa disciplina aborda o desenvolvimento de sistemas de software, focalizando a aplicação sistemática, disciplinada e quantitativa para o desenvolvimento, operação e manutenção de software;

- **Produto Integrado e Processo de Desenvolvimento** – essa disciplina objetiva a colaboração adequada entre as equipes de trabalho durante a vida do produto, para satisfazer melhor as necessidades do cliente, suas expectativas e requisitos;
- **Aquisição:** Esta disciplina abrange a aquisição de produtos e serviços de terceiros, ao invés de produzi-los, para serem usados no produto final que será produzido. A garantia da qualidade é realizada através do monitoramento das atividades dos fornecedores antes da entrega do produto.

O CMMI incorporou uma grande melhora com relação aos três modelos anteriores (SW-CMM [PALUK,1993], SECM [SECM,2007] e IPD-CMM [IPPD,2007]), pois possui duas formas de utilização. A forma tradicional, por estágios em níveis de maturidade e a forma contínua, muito parecida com a ISO/IEC 15504 [CHRISSIS,2003].

Na representação por estágios, o CMMI trabalha semelhantemente ao SW-CMM. Nessa representação, os níveis de maturidade estão classificados em 1-Inicial, 2-Gerenciado, 3-Definido Quantitativamente, 4-Gerenciado e 5-Otimizado (Figura 2.1).

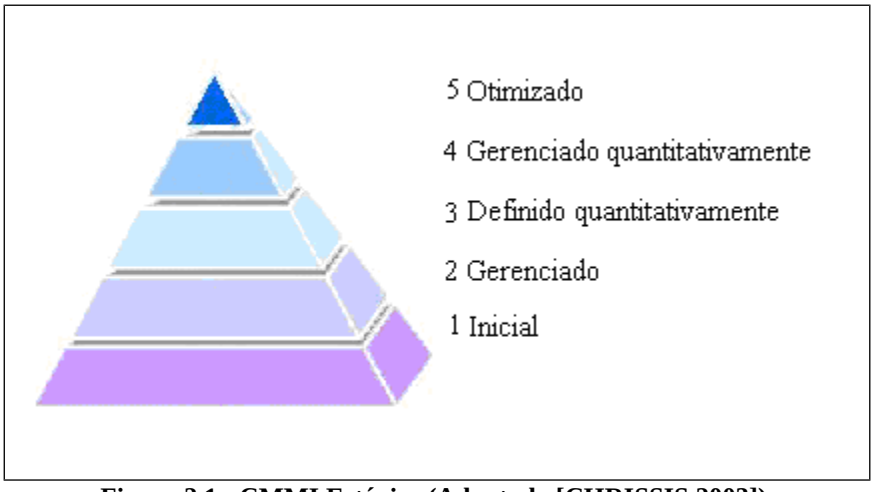


Figura 2.1 - CMMI Estágios (Adaptado [CHRISSIS,2003]).

Cada nível de maturidade define um conjunto de áreas de processo (AP), que especificam uma guia de melhoria para a organização. Para uma organização obter o certificado do nível almejado, ela deve implantar todas as APs relacionadas a ele. Na Tabela 2.1 são apresentadas as APs e seus respectivos níveis, com exceção do nível 1 que corresponde a uma fase inicial, em que ainda não há controle formal da qualidade, definida pelo CMMI como um nível caótico.

Tabela 2.1 - Áreas de Processo do modelo CMMI

Nível	APs
2: Gerenciado	• Gerenciamento de Requisitos. • Planejamento de Projeto. • Controle e Monitoração de Projeto. • Gerenciamento de Contrato de Fornecedores. • Medições e Análises. • Garantia de Qualidade de Produto e Processo. • Gerenciamento da Configuração.
3: Definido	• Desenvolvimento de Requisitos. • Solução Técnica. • Integração de Produto. • Verificação. • Validação. • Foco no Processo Organizacional. • Definição de Processo Organizacional. • Treinamento Organizacional. • Gerenciamento de Projeto Integrado. • Gerenciamento de Risco. • Resolução e Análise de Decisão.
4 : Quantitativamente Gerenciado	• Desempenho do Processo Organizacional. • Gerenciamento Quantitativo do Projeto.
5: Otimização	• Disposição e Inovação Organizacional. • Resolução e Análise Causal.

Cada área de processo é um conjunto de práticas que, quando executadas corretamente, satisfaz um conjunto de objetivos considerados importantes para implementar melhorias significativas na área de processo em questão [CHRISSIS,2003].

As APs são detalhadas pelas práticas específicas e pelas práticas genéricas. Essas práticas, conforme o CMMI [CMMI,2006], especificam o que deve ser

cumprido, definindo documentos, treinamentos ou políticas organizacionais para as atividades, porém não especificam como elas devem ser implementadas. Os objetivos específicos e genéricos são satisfeitos pela execução das práticas específicas e genéricas das áreas de processo (Figura 2.2).

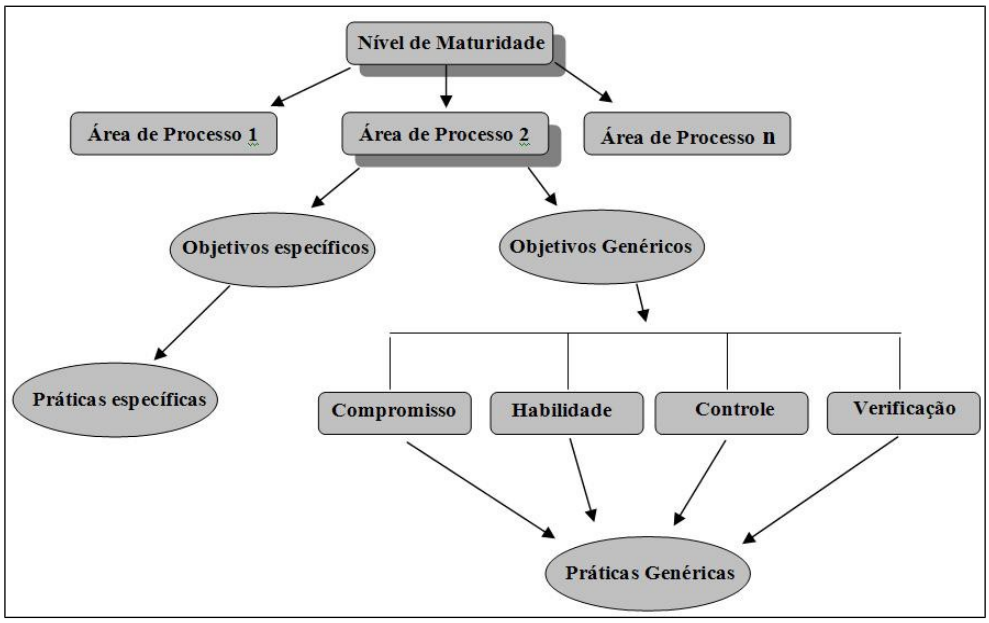


Figura 2.2 - Representação escalonada do CMMI (Adaptado [CMMI,2006])

A representação contínua oferece flexibilidade para o processo de melhoria. Uma organização pode escolher trabalhar com uma ou várias APs, que estão alinhadas com os objetivos do negócio. A representação também permite ter níveis diferentes em processos distintos de uma empresa. Por exemplo, ela pode desejar ter nível 2 em uma AP e nível 4 em outra. Assim, a organização pode direcionar seus esforços de melhoria nas áreas que julgar mais relevantes para o desenvolvimento como um todo.

Cada AP é dividida em seis níveis de capacidade, listados na Tabela 2.2:

Tabela 2.2 - Níveis de Capacidade das áreas de processo [CMMI,2006]

Nível	Descrição
0	Incompleto
1	Executado
2	Gerenciado
3	Definido
4	Gerenciado Quantitativamente

5	Otimizado

Nesta abordagem, assim como na abordagem em estágios, cada AP é composta por objetivos específicos e genéricos, organizados em práticas específicas e genéricas respectivamente, com exceção do Nível 0. Cada área de processo possui seus objetivos específicos próprios, compostos por suas práticas específicas. Os objetivos genéricos e as práticas genéricas são aplicados a múltiplas áreas de processo, descrevendo uma seqüência de níveis de capacidade. Cada nível possui um objetivo genérico, comum a todas as áreas no mesmo nível, como mostrado na Figura 2.3.

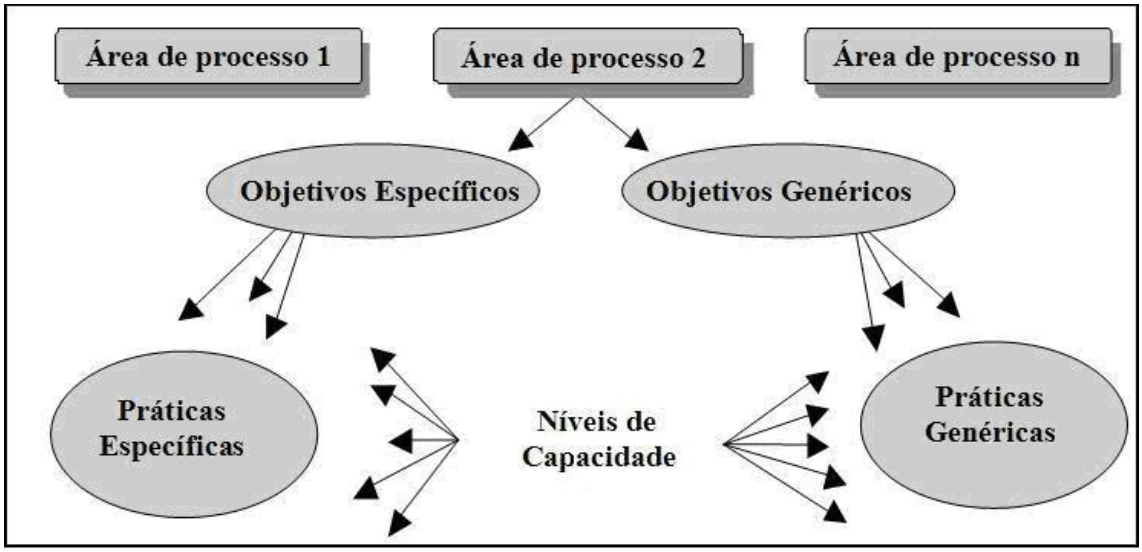


Figura 2.3 - Representação Contínua (Adaptado [CMMI,2006])

O modelo CMMI conta com várias APs que tratam de Gerenciamento de Projetos, objeto de estudo desta dissertação. Dentre elas destacam-se as seguintes:

- **Planejamento de Projeto:** compreende o estabelecimento e a manutenção de planos que definam as atividades do projeto.
- **Controle e Monitoração de Projeto:** proporciona um entendimento do processo utilizado em um projeto, tal que ações corretivas apropriadas possam ser tomadas quando o desempenho do projeto desvia significativamente do plano estabelecido.
- **Gerenciamento de Projeto Integrado:** tem por finalidade estabelecer e gerenciar o

projeto e o envolvimento das pessoas relevantes (indivíduos ou grupos envolvidos com o projeto, como fornecedores, clientes, usuários, e outros), de acordo com um processo definido e integrado baseado nos processos padrões da organização.

- **Gerenciamento de Riscos:** o objetivo dessa área é identificar potenciais problemas antes que eles ocorram, através do planejamento e execução de atividades específicas em situações de riscos, visando atenuar os impactos adversos que possam influenciar no alcance dos objetivos.
- **Gerenciamento Quantitativo do Projeto:** o propósito dessa área é gerenciar quantitativamente o processo definido para o projeto, visando atingir os objetivos de qualidade e de desempenho estabelecidos para o mesmo.

Para ilustrar como o CMMI aborda uma AP, apresenta-se como é tratada a área de "Planejamento de Projeto", com seus respectivos objetivos específicos, subdivididos nas suas práticas específicas como mostra a Tabela 2.3:

Tabela 2.3 - Objetivos e práticas específicas [CMMI,2006]

Objetivos específicos	Práticas específicas
1. Estabelecer estimativas	1. Estimar o escopo do projeto 2. Estabelecer estimativas dos produtos de trabalho e dos atributos das tarefas. 3. Definir o ciclo de vida do projeto 4. Determinar estimativas de esforço e custo
2. Desenvolver um plano de projeto	1. Estabelecer o orçamento e o cronograma 2. Identificar riscos do projeto. 3. Plano para gerenciamento dos dados. 4. Plano para recursos do projeto. 5. Plano para necessidades de conhecimento e especialidades. 6. Plano de envolvimento da equipe. 7. Estabelecer um plano de projeto.
3. Obter comprometimento para o plano	1. Revisão dos planos dos subordinados. 2. Ajustar o trabalho com os níveis de recursos.

	3. Obter plano de comprometimento.

O CMMI foi estudado nesta dissertação porque representa um dos modelos de qualidade mais utilizados no mundo e por ter sido a base para definição do modelo MPS.BR [MPSBR,2007] que está atualmente sendo seguido nas empresas abordadas nos estudos de caso. Conforme dados do Ministério da Ciência e Tecnologia [MCT,2007] observou-se que de 2002 a 2004 houve um crescimento expressivo de certificações no CMMI (de 9 certificações em 2002 para 49 em 2004); porém, de 2005 a 2006 não houve aumento nas certificações nas empresas brasileiras. Por outro lado, em 2004 criou-se o modelo de qualidade de software brasileiro MPS.BR [MPSBR,2007], que vem crescendo em número de certificações e atualmente já conta com 48 empresas.

2.3 MPS.BR

O MPS.BR é um programa para Melhoria de Processo do Software Brasileiro, coordenado pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), contando com apoio do Ministério da Ciência e Tecnologia (MCT), da Financiadora de Estudos e Projetos (FINEP) e do Banco Interamericano de Desenvolvimento (BID) e de diversas instituições de ensino e pesquisa.

O principal foco do programa MPS.BR é possibilitar a certificação de micro e pequenas empresas nacionais. O MPS.BR busca atender adequadamente ao contexto das empresas brasileiras, mantendo a compatibilidade com as abordagens internacionais na avaliação e melhoria de processo de software.

O MPS.BR possui três componentes:

- **Modelo de Referência para Melhoria de Processo de Software (MR-MPS)**: que contém os requisitos que deverão ser atendidos para estar em conformidade com o

Projeto MPS.BR. Ele contém as definições dos níveis de maturidade, da capacidade de processos e dos processos em si.

- **Método de Avaliação para Melhoria de Processo de Software (MA-MPS):** que contém o processo de avaliação, os requisitos para os avaliadores e os requisitos para averiguação da conformidade ao modelo MPS.BR.

- **Modelo de Negócio para Melhoria de Processo de Software (MN-MPS):** que contém uma descrição das regras para a implementação do MR-MPS pelas empresas de consultoria de software e de avaliação.

A base técnica utilizada para a construção do MPS.BR é composta pelas normas NBR ISO/IEC 12.207 – Processo de Ciclo de Vida de Software, a norma ISO/IEC 15.504 – Avaliação de Processo e, também, o CMMI-SE/SW [CMMI,2006], como mostrado na Figura 2.4:

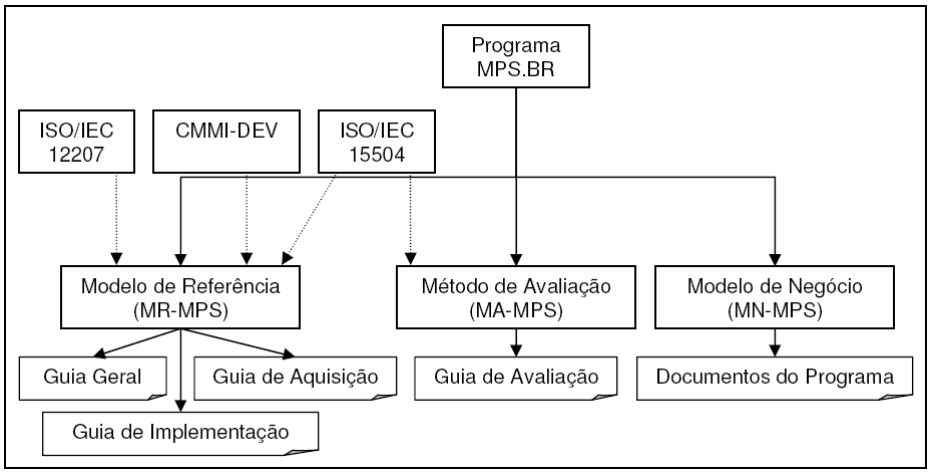


Figura 2.4 - Estrutura do MPS.BR [MPSBR,2007]

O Modelo de Referência MR-MPS define níveis de maturidade (Figura 2.5) que são uma combinação de processos (declarando os propósitos e resultados de sua execução) e capacidade de processo (habilidade para alcançar os objetivos de negócio, atuais e futuros).

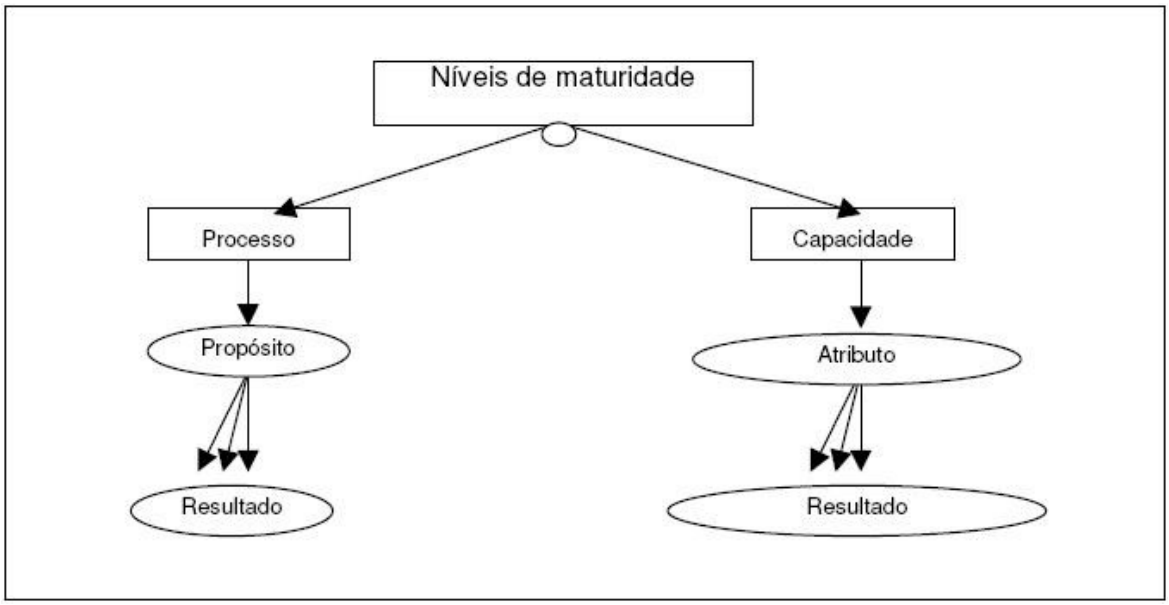


Figura 2.5 - Estrutura do MR-MPS (Adaptado de [MPSBR,2007])

Os níveis de maturidade estabelecem os níveis de melhoria de implementação de processo na organização. O MR-MPS define os níveis de maturidade com a escala que inicia no nível G e progride até o nível A, como mostra a Tabela 2.4:

Tabela 2.4 - Níveis de Maturidade do MR-MPS

Nível	Descrição
A	Em Otimização
B	Gerenciado Quantitativamente
C	Definido
D	Largamente Definido
E	Parcialmente Definido
F	Gerenciado
G	Parcialmente Gerenciado

Os níveis de maturidade foram baseados no modelo de qualidade CMMI-SE/SW, porém têm uma graduação diferente, que torna mais fácil e adequada a implementação e avaliação nas micro, pequenas e médias empresas de software. A possibilidade de se realizar avaliações considerando mais níveis permite obter resultados com prazos mais curtos que o CMMI-SE/SW.

No MPS.BR as condições necessárias para ascender a um nível de maturidade estão organizadas em atributos de processos, descritos na Tabela 2.5. Cada

nível de maturidade contém um conjunto de processos. Todo processo deve atender aos atributos de processo correspondentes aos níveis de maturidade, em que se encontra. A implantação da qualidade é acumulativa, ou seja, se a organização está no nível F, tem que atender ao nível de capacidade do nível G e do nível F.

A capacidade de um processo estabelece o grau de refinamento e institucionalização com que o processo é executado dentro da organização. À medida que evolui nos níveis, um maior ganho de capacidade para desempenhar o processo é atingido pela organização.

A capacidade do processo possui nove (9) atributos de processos (AP) que são: AP 1.1, AP 2.1, AP 2.2, AP 3.1, AP 3.2, AP 4.1, AP 4.2, AP 5.1 e AP 5.2;. Cada AP está detalhado em termos dos resultados para o alcance completo do atributo e estão listados na Tabela 2.5:

Tabela 2.5 - Níveis de capacidade do processo [MPSBR,2007]

Capacidade do processo	
Atributos de Processo	**Descrição**
AP 1.1	O Processo é executado
AP 2.1	O Processo é Gerenciado
AP 2.2	Os Produtos de trabalho do processo são gerenciados
AP 3.1	O Processo é definido
AP 3.2	O Processo está implementado
AP 4.1	O processo é medido
AP 4.2	O processo é controlado
AP 5.1	O processo é objeto de inovações
AP 5.2	O processo é otimizado continuamente

Os níveis de maturidade relacionados com a capacidade do processo estão apresentados na Tabela 2.6:

Tabela 2.6 - Nível de maturidade e nível de capacidade (Adaptado [MPSBR,2007])

Nível	Processo	Capacidade (Atributos de Processo)
G	Gerência de Requisitos –GRE	AP 1.1 e AP 2.1
	Gerência de Projetos – GPRs	
F	Medição – MED	AP 1.1, AP 2.1 e AP 2.2
	Garantia da Qualidade – GQA	
	Gerência de Configuração – GCO	
	Aquisição – AQU	
E	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Gerência de Reutilização – GRU	
	Gerência de Recursos Humanos – GRH	
	Definição do Processo Organizacional – DFP	
	Avaliação e Melhoria do Processo Organizacional – AMP	
D	Verificação – VER	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Validação – VAL	
	Projeto e Construção do Produto – PCP	
	Integração do Produto – ITP	
	Desenvolvimento de Requisitos – DRE	
C	Gerência de Riscos – GRI	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Desenvolvimento para Reutilização – DRU	
	Análise de Decisão e Resolução – ADR	
	Gerência de Reutilização – GRU (evolução)	
B	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2,AP 3.1 e AP3.2, AP 4.1 e AP 4.2
A	Análise de Causas de Problemas e Resolução – ACP	AP 1.1, AP 2.1, AP 2.2, AP 3.1, AP3.2, AP 4.1, AP 4.2 , AP 5.1 e AP 5.2

Os objetivos estabelecidos no processo Gerência de Projetos do Nível G, serão analisados detalhadamente no Capítulo 5, observando os benefícios obtidos pelo emprego da estratégia aqui proposta, nas as empresas que estão implantando o MPS.BR.

2.4 PSP - Processo Pessoal de Software

O PSP é um processo que tem por objetivo melhorar a previsibilidade, a produtividade e, principalmente, a qualidade do trabalho de um desenvolvedor de software [HUMPHREY,1995]. Embora o PSP use vários métodos convencionais de engenharia de software, seu principal objetivo é mostrar aos desenvolvedores que um processo bem definido e controlado pode ajudá-los a melhorar seu desempenho pessoal. O PSP consiste em sete níveis que, gradualmente, introduzem novos dados e análises que, por sua vez, podem ser utilizados para determinar o desempenho e medir a eficácia dos métodos utilizados.

As práticas do PSP são derivadas do CMM (*Capability Maturity Model*) [PAULK,1993], também desenvolvido pelo SEI (*Software Engineering Institute*), e precursor do CMMI [CMMI,2007], apresentado anteriormente. O CMM captura as melhores práticas de desenvolvimento de software utilizadas por grandes corporações e é organizado em cinco níveis de maturidade. O PSP é o resultado de uma adaptação do CMM, mas com foco no indivíduo. O PSP reconhece que cada *desenvolvedor* tem características únicas e assim, esse processo deve ser adaptado à maneira de trabalho de cada um. Ao usar os dados imediatamente disponibilizados pelo PSP, os Desenvolvedores podem determinar quais métodos são mais eficazes em seu modo de trabalho.

O PSP sugere que as atividades de desenvolvimento de software sejam divididas em Planejamento, Desenvolvimento (projeto, codificação e testes) e Análise Final, embora variações sejam aceitas. Anteriormente ao planejamento, assume-se que o

levantamento de requisitos tenha sido realizado e que o artefato gerado desse levantamento sirva como entrada à primeira etapa mencionada.

Durante o Planejamento, todas as estimativas são realizadas e o desenvolvedor se concentra em produzir valores que se aproximem ao máximo dos futuros dados reais. Durante o Desenvolvimento, o desenvolvedor despende a maior parte do tempo efetivamente construindo o software, ou seja, projetando, codificando e testando o software. Por último, como já diz o próprio nome, a Análise Final é utilizada para levantar informações a respeito de todo o trabalho realizado nas fases anteriores e estabelecer um comparativo entre o que foi planejado e estimado com o que foi executado realmente.

O **Nível 0**, chamado de *Baseline*, representa apenas uma pequena introdução do PSP nas práticas de desenvolvimento pessoal. Nessa etapa, as práticas do PSP são abrangentes o suficiente para que o desenvolvedor sinta pouca diferença entre o processo de desenvolvimento que ele já usa e o que está sendo introduzido. Algumas métricas começam a ser coletadas para que os alicerces do processo de melhoria sejam estabelecidos, como o tempo gasto no desenvolvimento e o número de defeitos identificados em cada fase de desenvolvimento.

No **Nível 0.1** são acrescentados os conceitos de padronização de código e de contagem de linhas de código. Basicamente, o desenvolvedor define um mínimo de estimativas na fase de planejamento, indicando quanto tempo ele pretende dedicar para realizar cada fase do processo, quantas linhas de código serão criadas, alteradas e reusadas, para então iniciar a fase de desenvolvimento. Cada defeito identificado durante o processo é registrado e armazenado em um *Log* de Defeitos, para que se consiga identificar, exatamente, as maiores dificuldades.

O **Nível 1** acrescenta atividades de planejamento ao PSP. Antes de iniciar um projeto com PSP1, pode-se usar um método de estimativa chamado PROBE para estimar a quantidade de linhas de código novas ou modificadas no novo programa.

O PROBE é um método algorítmico para estimativa de tamanho que utiliza um banco de dados histórico para determinar uma tendência na relação entre o tamanho calculado e o tamanho atual, assumindo que o desempenho do passado é um indicativo do desempenho futuro. A qualidade da estimativa deve melhorar depois que é compreendida a utilização dos dados históricos, proporcionando ao *desenvolvedor* estimativas melhores que compensem as tendências pessoais, ou seja, se o desenvolvedor sempre estima para menos, ele percebe essa tendência e efetua a correção.

O relatório de testes, juntamente com estimativas de tempo e recursos, são os principais acréscimos desse nível. Assim, possibilita-se que desenvolvedores comecem a entender a relação entre linhas de código produzidas e o tempo despendido no desenvolvimento, ao mesmo tempo que aprendem como planejar e avaliar seu trabalho com os dados coletados.

O objetivo do **Nível 1.1** é acrescentar o planejamento de recursos (principalmente tempo), à comparação do desempenho do desenvolvedor frente a suas estimativas, e os formulários de planejamento de tempo e cronograma. Isso significa que, a partir desse nível, o desenvolvedor realiza, na fase de Planejamento, as estimativas de quantidade de linhas de código e de tempo a ser despendido em cada fase de desenvolvimento. Dessa forma, é possível organizar o trabalho e medi-lo freqüentemente ao longo de todo ciclo de desenvolvimento. Isso mostra ao desenvolvedor como é o desempenho de seu trabalho em cada uma das fases do processo, possibilitando que se notem os pontos fracos e fortes, e que assim o desenvolvedor procure aprimorar os aspectos que representam maiores problemas para

o seu desempenho pessoal.

A qualidade passa a ser o principal critério do **Nível 2**. O ponto mais importante é a introdução de *checklists* de revisão usados para a identificação precoce de defeitos. Inicialmente, esses *checklists* são próprios do PSP, mas o desenvolvedor é encorajado e responsável por analisar seus pontos fracos no desenvolvimento e, assim, personalizar os *checklists*.

No **Nível 2.1** a preocupação não é só com a qualidade da implementação do software, mas também com a qualidade com que o projeto do sistema é conduzido. Dessa forma, o PSP não diz exatamente como deve ser feita a modelagem ou a especificação do software, mas sugere como fazer especificações completas e consistentes. O objetivo é ajudar a reduzir os defeitos inseridos, principalmente, na fase de projeto, oferecendo critérios para a avaliação dos mesmos.

Até os níveis anteriormente descritos, o PSP é praticável para programas com algumas poucas mil linhas de código. No caso do software ser muito grande, é muito provável perder a lógica do sistema ou despender muito tempo em testes, considerando o software como uma única unidade. O **Nível 3** possui o objetivo de ajudar o desenvolvedor na divisão de seu projeto de muitas mil linhas de código em pequenos projetos adequados ao PSP Nível 2.1. Para isso, o PSP propõe que uma porção principal do software seja desenvolvida com a aplicação dos níveis 2 e 2.1, e que as diversas iterações necessárias sejam executadas sobre esse "*kernel*". A cada ciclo de iteração, os critérios de qualidade são aplicados garantindo-se que outra iteração possa ser realizada sem que ocorra regressão no desenvolvimento. Assim, o PSP mostra-se como um processo bem definido de melhoria contínua pessoal, útil tanto para pequenos sistemas quanto para sistemas maiores.

Para facilitar o uso do PSP várias ferramentas foram propostas. Na seção

seguinte comenta-se sobre esse apoio dando-se ênfase à ferramenta *Process Dashboard* [DASHBOARD,2007] que é utilizada pelas empresas mencionadas neste trabalho.

2.4.1 - *Ferramenta Process Dashboard*

As ferramentas que apóiam o uso do PSP são classificadas em:

• 1ª. Geração: que corresponde ao uso de formulários em papel e cronômetros para apoiar todas as atividades do processo. Essa primeira geração de ferramentas foi proposta por Watts Humphrey juntamente com o PSP em seu livro "*A Discipline for Software Engineering*". São fornecidos vários *templates* que apóiam as várias atividades do PSP em seus sete níveis. Essas ferramentas caracterizam-se por serem extremamente trabalhosas de se usar.

• 2ª Geração: ferramentas computacionais de apoio ao PSP. Essas ferramentas auxiliam o desenvolvedor elaborando cálculos e gráficos, mas ainda é tarefa do desenvolvedor fornecer os dados (número de linhas, número de defeitos, etc). Neste caso se enquadra a ferramenta *Process Dashboard* [DASHBOARD,2007], usada no contexto deste trabalho, pois é adotada nas duas empresas abordadas nos estudos de casos.

• 3ª Geração: essas ferramentas caracterizam-se como sensores inteligentes capazes de funcionar sem a intervenção direta do desenvolvedor. Geralmente se "acoplam" à IDE (*Integrated Development Environment*)s como Eclipse e Netbeans, capturando os dados de uso. São eficazes na coleta de dados de tempo e tamanho, geralmente obtendo resultados mais precisos do que as ferramentas de segunda geração. No entanto, nem todas as atividades do PSP podem ser apoiadas por este tipo de ferramenta, uma vez que é impossível automatizar a coleta de todos os tipos de métricas.

A ferramenta *Process Dashboard* [DASHBOARD,2007], foi escrita em *Java* para *Desktop* com o conceito de código aberto, o que permite que ela seja executada da mesma forma tanto em plataformas Linux quanto em *Windows*. Com ela é

uma ferramenta de 2ª geração, o desenvolvedor deverá informar quando vai iniciar uma tarefa, quando terminou e as interrupções. Para tanto, basta apenas um *click* do *mouse* no botão de controle da ferramenta, que fica posicionado na barra de ferramentas do *Windows*, geralmente no canto direito inferior ao lado do relógio.

Os sete níveis do PSP, descritos na seção anterior, estão contemplados na ferramenta, e para cada um dos níveis, a ferramenta fornece formulários, especificados nos documentos do processo PSP [HUMPHREY,1995], para que o *desenvolvedor* preencha com informações de seu processo pessoal de Desenvolvimento. Esses formulários servem como uma guia que auxilia o *desenvolvedor* a executar as atividades do PSP.

Pela facilidade da ferramenta em gerar gráficos e relatórios, o custo da atividade *Postmortem* se torna baixo e diminui a resistência, por parte dos Desenvolvedores, em executarem o processo PSP. Além dessa facilidade, a ferramenta ainda conta com mecanismos que auxiliam o *desenvolvedor* a contar as linhas de código, registrar o tempo de cada atividade, registrar os defeitos informando tipo e tempo em cada atividade prevista no PSP para solucionar os defeitos.

Na Figura 2.6, observa-se a interface da ferramenta e na Tabela 2.7, descreve-se a funcionalidade de cada botão.

Existe também a possibilidade de registrar o tempo de execução de tarefas que não são codificação de software, como documentação, por exemplo. Nesses casos não há como estimar tamanho; apenas a estimativa do tempo é feita. O registro dessas tarefas ajuda a controlar o projeto, e os dados históricos fornecem base para planejamento de novos projetos.

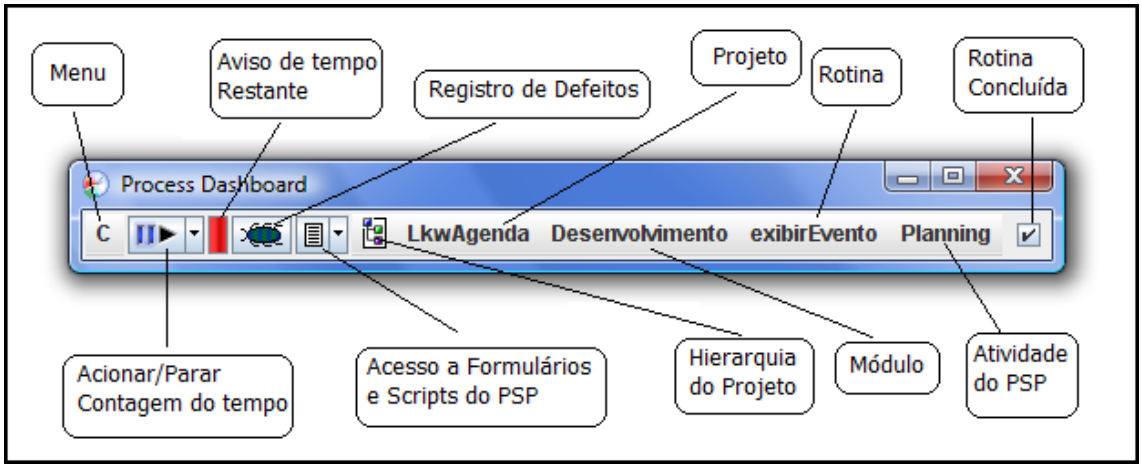


Figura 2.6 - Visão dos botões da ferramenta *Process Dashboard*

Tabela 2.7 - Funcionalidades de cada botão do *Process Dashboard*

Operação do Menu	Descrição
Menu	Dá acesso às funções de análise de dados e configurações.
Acionar/Pausar contagem de tempo	Inserir e defeitos e marcar o tempo para sua resolução
Aviso de Tempo Restante	Uma barra representa graficamente o tempo que falta para concluir uma rotina em relação ao tempo estimado. Quando o tempo real ultrapassa o tempo estimado a barra fica vermelha.
Registro de Defeitos	Registra os defeitos, tipo de erro, atividade em que ele foi injetado, atividade em que ele foi resolvido.
Acesso a formulários e scripts do PSP.	Formulários e Scripts definidos no documento da SEI [HUMPHREY,2007]
Hierarquia do Projeto	Navegação por projeto ou por rotinas.
Acesso aos projetos	Acesso a todos os projetos. A estrutura dos projetos, nesta ferramenta, se assemelha com as ferramentas de navegação em diretórios de um sistema de arquivos.
Módulo	Navegação nos módulos do sistema.
Rotina	Navegação nas rotinas do sistema
Fases de desenvolvimento previstas na metodologia PSP.	No caso do PSP 1.1, temos a Analise, Projeto, Codificação, Compilação, Teste e *Postmorten*
Tarefa Concluída	Marcar que a tarefa foi concluída.

Na Figura 2.7, mostra-se a interface da ferramenta com o registro dos tempos de cada atividade prevista no PSP (*Planning*, *Designer*, *Code*, *Compile*, *Test*, *Postmortem*) de um projeto. Em destaque na Figura 2.7, observa-se que o módulo “LwkAgenda” foi desenvolvido em 53:17h, e que a rotina “ExibirEvento” que está em

destaque, foi desenvolvida em 14:36h. Essa apresentação permite ao desenvolvedor uma visão de seu trabalho como um todo e também uma visão detalhada de uma rotina específica. À direita são apresentados os registros de tempo em cada atividade do PSP, e suas interrupções. Pode-se filtrar os tempos por período e inserir comentários para cada registro.

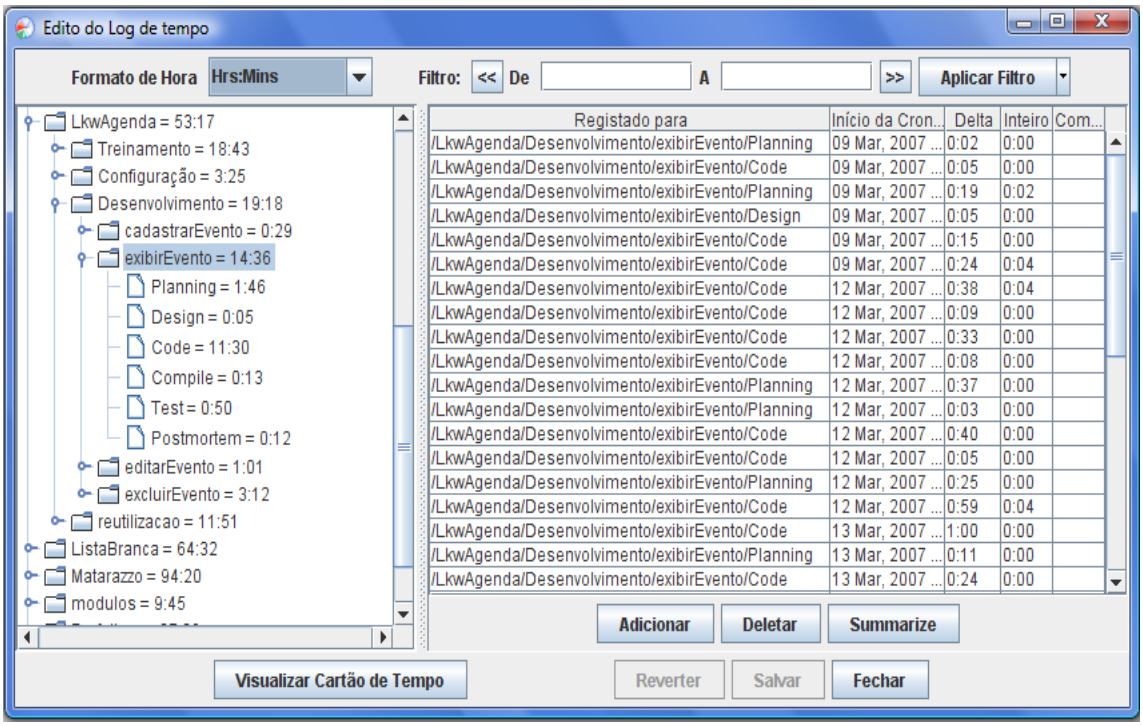


Figura 2.7 - Registros dos tempos

Na Figura 2.8, mostram-se alguns gráficos que a ferramenta *Dashboard* oferece ao *desenvolvedor*. Nessa figura, observam-se doze tipos de gráficos diferentes, selecionados a partir do menu da esquerda, no qual estão relacionados diversos gráficos e relatórios. Caso o *desenvolvedor* tenha a necessidade de algum relatório que não consta na ferramenta, é possível elaborar uma representação gráfica, ou um relatório personalizado.

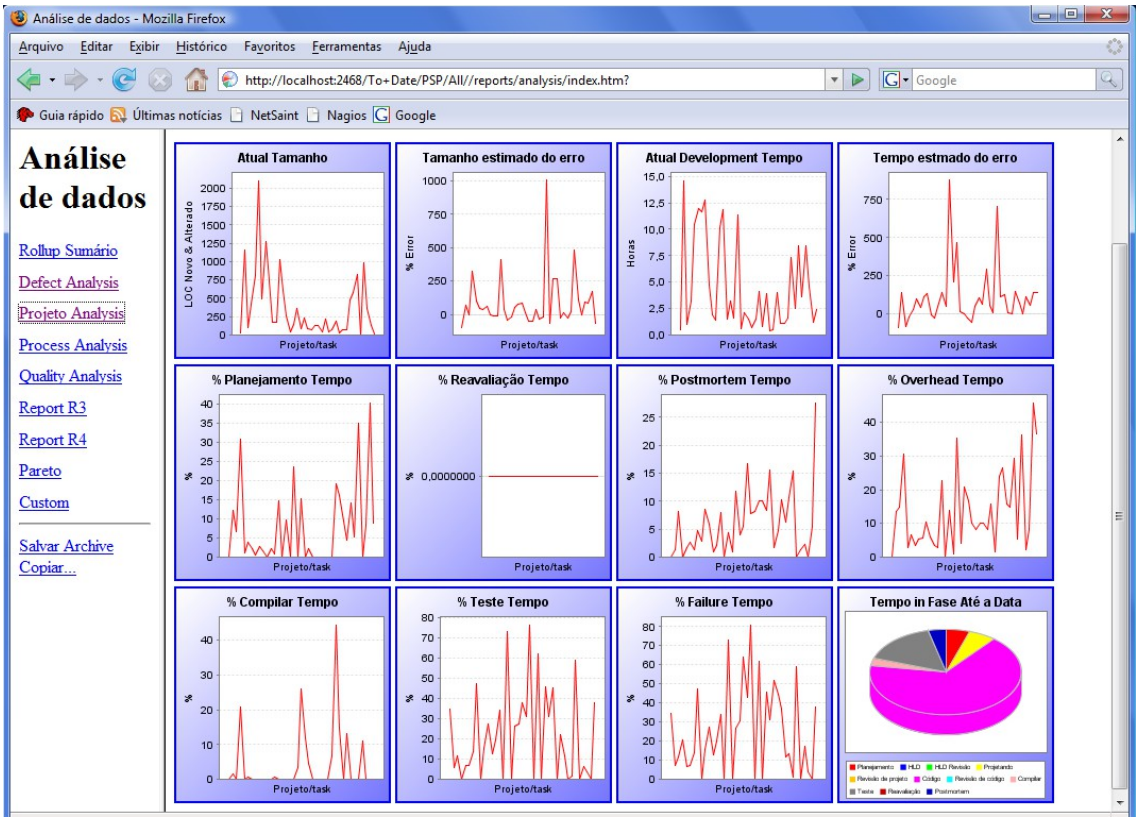


Figura 2.8 - Gráficos da Ferramenta *Process Dashborad*

Essa ferramenta apresenta a desvantagem de depender totalmente do *desenvolvedor* acionar o registro do tempo de suas tarefas. Estuda-se, para trabalhos futuros, a substituição desta ferramenta para uma que faça o registro dos tempos e defeitos, automaticamente.

2.5 – Considerações Finais

Neste capítulo foram apresentados os modelos de qualidade de processo CMMI e MPS.BR bem como o PSP, que é um modelo de qualidade de processo pessoal. Como foi mencionado, tanto o CMMI como o PSP têm sua base no CMM e, por sua vez, o MPS.BR tem sua base no CMMI.

No contexto deste trabalho esses modelos têm sua importância da seguinte forma: o PSP é utilizado como ferramenta indispensável para a estratégia aqui proposta, pois é pela aplicação desse processo que os desenvolvedores dão a

realimentação necessária para a estratégia manter-se ativa e funcionando. Em relação ao CMMI e MPS.BR, a estratégia aqui proposta aborda um item fundamental para ambos, que é o gerenciamento de projetos, podendo ser uma alternativa para implantação dos requisitos relacionados a esse item.

Salienta-se que a estratégia aqui proposta foi definida no contexto das empresas tratadas nos estudos de caso apresentados no Capítulo 6 e que nessa empresas, já há bastante tempo existe uma determinação para melhoria de processo, melhoria esta que se iniciou com base no CMM, passando para o CMMI e agora para o MPS.BR, uma vez que este último é um modelo mais compatível ao porte das referidas empresas.

No próximo capítulo, serão tratados alguns conceitos específicos sobre gerenciamento de projetos, que é um assunto diretamente relacionado com o trabalho.

Capítulo 3 – Planejamento de Software

3.1 Considerações iniciais

Um dos aspectos fundamentais do planejamento e gerenciamento de projetos é a compreensão de quanto o projeto provavelmente custará. Estimativas de custo elevadas podem fazer com que os clientes cancelem projetos, ao passo que uma estimativa de custo abaixo do real pode forçar a equipe do projeto a investir muito do seu tempo, sem conseguir compensação financeira. Uma boa estimativa de custos, feita logo no começo do projeto, ajudará o *gerente* do projeto a saber quantos desenvolvedores serão necessários e a conseguir que o pessoal adequado às características do projeto esteja disponível quando necessário.

Neste capítulo são apresentadas as principais atividades para construir um planejamento de software de forma tradicional. Como a estratégia de gerência de projeto proposta neste trabalho promove a agilidade e se enquadra muito naturalmente ao processo do método ágil *Scrum*, no Capítulo 4 serão abordados os conceitos dos Métodos Ágeis, com atenção particular ao planejamento ágil.

Este capítulo está organizado da seguinte forma: a seção 3.2, apresenta a métrica APF (Análise por Pontos de Função), e o método PCU (Pontos por Caso de Uso). A métrica APF foi revisada neste capítulo por ser a mais conhecida, e ser base para o método PCU, que foi utilizado nesta dissertação. Na seção 3.3, faz-se uma revisão sobre a gerência de riscos, na seção 3.4 é apresentada, sucintamente, a construção de um plano de projeto de software e, finalmente, na seção 3.5 são apresentadas as considerações finais.

3.2 Métodos de apoio ao planejamento de software

Para se fazer o planejamento de software é necessário ter uma previsão do seu tamanho, sendo que esse conceito pode estar vinculado tanto ao número de linhas de código que esse software provavelmente terá, como também pode estar associado a um valor mais abstrato que retrate a complexidade de implementá-lo. De qualquer forma, mesmo que seja usada uma métrica que retrate a complexidade do software, em geral, associa-se essa complexidade ao tamanho do software. Assim, nesta seção, serão abordadas duas técnicas que prevêem a complexidade, mas cujo valor, na prática, caracteriza o tamanho do software a ser implementado.

3.2.1 Pontos por Função

A APF (Análise por Pontos de Função) foi criada em 1979 por Allan Albrecht [ALBRECHT,1979] e, mais tarde, refinada pelo *International Function Point Users Group* (IFPUG). Essa métrica permite uma contagem indicativa do tamanho do software no início do desenvolvimento, sem a necessidade de se conhecer os detalhes do modelo de dados. Quando se realiza uma contagem de Pontos por Função (PF), todos os requisitos funcionais conhecidos são analisados, ponderados e contados.

A APF também fornece suporte para verificação dos requisitos não-funcionais que, apesar de terem um forte impacto no projeto como um todo, normalmente são avaliados em estágios posteriores do ciclo de vida de desenvolvimento. Analisando-se as 14 Características Gerais do Sistema (CGS) que o método define, é possível abstrair informações importantes acerca dos requisitos não-funcionais em estágios iniciais do projeto. Esses requisitos incluem aspectos de segurança, disponibilidade, desempenho, usabilidade, entre outros.

O método de APF envolve as seguintes atividades [ALBRECHT,1979]:

- Contagem dos 5 componentes lógicos que afetam de maneira distinta o tamanho do software: “Arquivos Lógicos Internos”, “Arquivos de Interface Externa”, “Consultas Externas”, “Saídas Externas” e “Entradas Externas”.
- Cálculo do valor do ponto de função não-ajustado com base na complexidade de cada um dos componentes lógicos.
- Cálculo do fator de ajuste do valor por meio da CGS e determinação do valor do ponto de função ajustado.

O método da APF considera dois tipos de funções a serem contadas: tipo **dado** e tipo **transação**. A fronteira da aplicação (Figura 3.1), e os componentes lógicos da primeira categoria são descritos detalhadamente abaixo:

- Arquivo Lógico Interno (ALI): é um grupo de dados logicamente relacionados ou informações de controle, identificados e modificados pelo usuário e mantidos dentro da fronteira da aplicação que está sendo medida.
- Arquivo de Interface Externa (AIE): é um grupo de dados logicamente relacionados ou informações de controle identificáveis pelo usuário e referenciados pela aplicação que está sendo analisada, mas mantidos dentro da fronteira de outra aplicação. Um AIE contado para uma aplicação deve ser contado como um ALI de uma outra aplicação.

Após a identificação dos ALI’s e AIE’s, determina-se a complexidade de cada tipo funcional, baseada na contagem dos seguintes tipos elementares:

- Dado Elementar Referenciado (DER): é um campo único não-repetido que está presente em um ALI ou AIE.
- Registro Lógico Referenciado (RLR): é um subgrupo único de elementos de dados não-repetido e reconhecido pelo usuário dentro de um ALI ou AIE. Representa cada chave presente no ALI ou AIE.

Já as funções do tipo transação englobam os seguintes componentes lógicos:

- Entrada Externa (EE): é qualquer função ou transação que leva dados ou informações de controle de fora para dentro da fronteira da aplicação.
- Saída Externa (SE): é qualquer função ou transação que fornece dados ou informações de controle para fora da fronteira da aplicação, cujo objetivo principal é a apresentação de informações ao usuário através de lógica de processamento, a qual deve conter, no mínimo, uma fórmula ou cálculo matemático, ou dados derivados.
- Consulta Externa (CE): é qualquer função ou transação que fornece dados ou informações de controle para fora da fronteira da aplicação, cujo objetivo principal é a apresentação de informações ao usuário através de recuperação de dados ou informações de controle de um ALI ou AIE; a lógica de processamento não deve conter fórmulas ou cálculos matemáticos, nem a criação de dados derivados.

Figura 3.1 - Fronteira da aplicação (adaptado de [ALBRECHT,1979])

Após a identificação das EE's, SE's e CE's, deve-se determinar a complexidade de cada tipo funcional, baseada na contagem dos seguintes tipos elementares:

- Dado Elementar Referenciado (DER): é um campo único não-repetido visível ao usuário.
- Arquivo Lógico Referenciado (ALR): é um ALI lido ou mantido por uma função transacional, ou um AIE lido por uma função transacional.

Baseando-se no grau de complexidade dos tipos funcionais, obtém-se o

valor de Pontos de Função Não Ajustados (PFN) para cada elemento funcional. Os graus de complexidade, ou pesos, podem ser considerados como: Baixo (W_B), Médio (W_M), ou Alto (W_A).

O valor de PFN de cada componente lógico é calculado multiplicando-se o número de elementos encontrados para cada complexidade pelo respectivo número de pontos de função equivalente. Por fim, o valor total de PFN é calculado somando-se os valores de PFN's obtidos para cada componente lógico.

Assim, o PFN total será calculado pela fórmula:

$$PFN = PFN_{ALI} + PFN_{AIE} + PFN_{EE} + PFN_{SE} + PFN_{CE}. \quad (1)$$

O valor de PFN pode ser considerado como a primeira estimativa válida de Pontos de Função. Seu valor poderá ainda ser ajustado para refletir fatores do ambiente e fatores técnicos que influenciam, de alguma forma, no tamanho funcional da aplicação.

O Fator de Ajuste do Valor (FAV) é baseado em 14 Características Gerais do Sistema (CGS), que medem o Grau Total de Influência (GTI) do ambiente. Cada característica está associada a uma descrição que auxilia a determinação do grau de influência. Um peso, que varia de 0 (sem influência) a 5 (forte influência), deve ser atribuído a cada característica e a soma deles resulta no valor do GTI. Por exemplo, a "comunicação de dados" trata do grau com que a aplicação se comunica com o processador. As 14 características são resumidas no FAV que, quando aplicado, corrige o valor de UFP em cerca de 35%, produzindo o valor de Pontos de Função Ajustado (PFA).

A Figura 3.2 exibe sucintamente todo o processo de contagem analisado na APF.

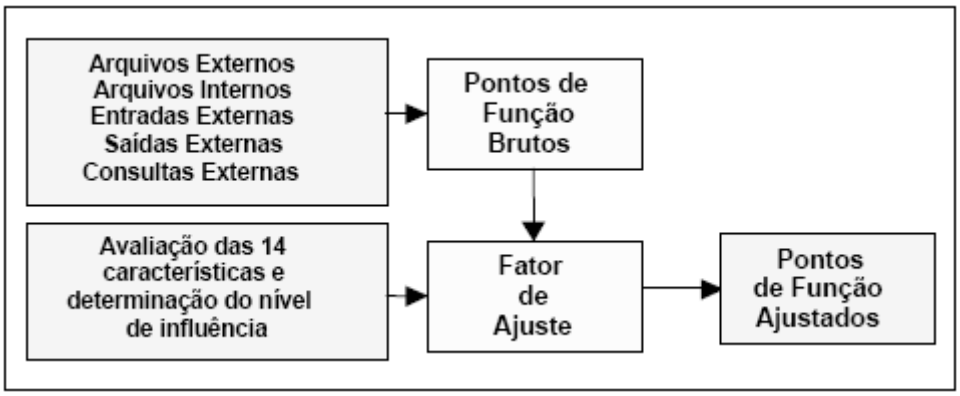


Figura 3.2 - Visão Geral do Processo de APF (adaptado de [ALBRECHT,1979]).

O método de APF pode ser usado nos mais variados sistemas porém, devido à sua complexidade, outros métodos foram desenvolvidos com o objetivo de serem mais simples e diretos na estimativa de esforço de desenvolvimento. Na próxima seção será apresentado o método de Pontos de Caso de Uso, desenvolvido por Karner [KARNER,1993], com o objetivo de estimar o porte dos sistemas orientados a objetos.

3.2.2 Pontos por Caso de Uso

Estimativas de custo e de tempo de produção de um sistema de software sempre representaram um desafio para os desenvolvedores, uma vez que não é uma tarefa trivial calcular o porte e, conseqüentemente, o custo do sistema. Diversas técnicas foram formuladas para facilitar a estimativa de esforço na construção de um sistema, entre as quais se destaca, o método de Pontos por Caso de Uso.

Esse método, proposto por Gustav Karner [KARNER,1993], foi baseado na métrica de APF (Análise por Pontos de Função) [ALBRECHT,1979]. As estimativas são calculadas tomando como base a complexidade dos Casos de Uso que compõem o sistema e dos Atores que utilizarão o mesmo. Essa estimativa também leva em consideração os Fatores de Complexidade Técnica relativos aos requisitos não funcionais, de forma semelhante à métrica APF. No entanto, acrescenta-se uma inovação proposta por Karner: a aplicação de Fatores Ambientais, que consideram o

nível de competência da equipe que desenvolverá o sistema [KARNER,1993].

A base para o cálculo da estimativa é a complexidade dos Casos de Uso e a complexidade dos Atores, que podem ser classificados como Simples, Médio ou Complexo. Os ajustes são feitos através dos Fatores Ambientais e dos Fatores de Complexidade Técnica.

O peso dos Atores é determinado de acordo com a interface disponível para utilização do sistema. Se o Ator acessa o sistema através de um outro sistema, por uso de uma API (*Application Program Interface*), é considerado simples e tem peso 1; se o sistema é acessado por uma interface texto ou interface gráfica, a complexidade do ator é, respectivamente, média ou complexa, cujos pesos são 2 e 3.

Os Casos de Uso são classificados de acordo com o número de transações atômicas ou pela quantidade de objetos de análise que foram identificados na fase de análise do sistema. Analogamente aos Atores, são classificados como simples, médios ou complexos. A Tabela 3.1 apresenta a classificação dos Casos de Uso, segundo o trabalho de Karner. Na Tabela 3.1, vê-se que um Caso de Uso pode ser considerado simples se o número de transações atômicas não ultrapassar a quantidade de três ou então, o número de objetos de análise não ultrapassar a quantidade de cinco.

A soma dos pesos de todos os Casos de Uso com os pesos de todos os Atores constitui os Pontos por Caso de Uso Não Ajustados.

Os Fatores de Complexidade Técnica (FCT) e os Fatores Ambientais (FA) são calculados para que se considerem os requisitos não funcionais e ambientais, respectivamente. O FCT foi adaptado da técnica de Pontos por Função, tendo alguns fatores adicionados, removidos ou alterados, de acordo com a experiência de Karner quando trabalhava na empresa *Objectory*. O FA é uma criação de Karner para a técnica de Pontos por Caso de Uso [KARNER,1993].

Tabela 3.1 - Classificação de Caso de Uso [KARNER,1993]

Tipo do Caso de Uso	Número de Transações	Número de Objetos de Análise	Peso
Simples	Até 3	Até 5	5
Médio	De 4 até 7	De 6 até 10	10
Complexo	Acima de 8	Acima de 11	15

Cada FCT e FA possuem diferentes pesos no desenvolvimento do sistema, como mostrado na Tabela 3.2. Atribui-se para cada FCT e FA um nível de influência que varia em uma escala de zero a cinco, sendo que zero significa que é irrelevante, enquanto cinco significa que é essencial. Se o fator não é essencial e nem irrelevante, Karner sugere que seja atribuído valor três, pois assim, se todos os fatores tiverem esse valor, o FCT e/ou FA será aproximadamente de valor um e, portanto, não influencia no ajuste dos Pontos por Casos de Uso.

Tabela 3.2 - Fatores de Complexidade Técnica e Fatores Ambientais [KARNER,1993]

FCT	Descrição	Peso	FA	Descrição	Peso
F1	Sistema distribuído	2	E1	Familiaridade com o processo de desenvolvimento.	1,5
F2	Tempo de Resposta	1	E2	Desenvolvedores em meio expediente.	-1
F3	Eficiência	1	E3	Presença de analistas experientes	0,5
F4	Processamento complexo	1	E4	Experiência com a aplicação em desenvolvimento.	0,5
F5	Código reusável	1	E5	Experiência em Orientação a Objetos.	1
F6	Facilidade de instalação	0,5	E6	Motivação	1
F7	Facilidade de uso	0,5	E7	Dificuldade com a linguagem de programação	-1
F8	Portabilidade	2	E8	Requisitos estáveis	2
F9	Facilidade de mudança	1			
F10	Concorrência	1			
F11	Recursos de segurança	1			
F12	Acessível por terceiros	1			
F13	Requer treinamento especial	1			

Os Fatores de Complexidade Técnica e os Fatores Ambientais são calculados de acordo com as seguintes fórmulas [KARNER,1993]:

$$FCT = 0,6 + 0,01\sum_{i-1}^{13} F_i * Influência_i \qquad FA = 1,4 - 0,03\sum_{i-1}^{8} E_i * Influência_i$$

sendo que F_i e E_i são, respectivamente, os pesos tabelados de cada FCT e FA, e a $Influência_i$ é o valor entre zero e cinco atribuído.

O Cálculo do total de Pontos por Caso de Uso de um sistema é realizado através da seguinte fórmula [KARNER,1993]:

PCU = Pontos por Caso de Uso não ajustados * FCT * FA

O valor do nível de esforço em horas/homens proposto por Karner é de 20 horas para cada Ponto por Caso de Uso [KARNER,1993]. Convencionou-se, nesta dissertação, chamar esse valor de NDE – Nível De Esforço. Por exemplo, um sistema que possua três casos de uso com complexidade média de 10 pontos cada, totalizando 30 pontos, dois atores complexos de 3 pontos cada, totalizando 6 pontos e com os seguintes Fatores Ambientais (FA): Familiaridade com o processo de desenvolvimento com valor em 5, e Experiência em Orientação a Objetos com valor de 5, e os outros fatores com valor de 3, totalizando o FA em 0,85. Os Fatores de Complexidade Técnica (FCT) definidos da seguinte forma: “Eficiência”, e “Código deve ser reutilizável” com valor 5, e os demais fatores com valor 3, totalizado o FCT em 1,06. Neste caso, o sistema teria um total de PCU=(30+6)*0,85*1.06 = 32,44 pontos por Caso de Uso. Considerando o valor proposto por Karner de 20 horas homem, esse sistema seria construído em aproximadamente 648 horas. Observa-se, que outros autores defendem valores diferentes do proposto por Karner. Agiar [AGUIAR,2006], por exemplo, defende um número entre 15 e 20 horas homem.

Segundo Probasco [PROBASCO,2002], a principal dificuldade encontrada para aplicar a técnica proposta por Karner está na compreensão da complexidade dos Casos de Uso. Probasco faz algumas recomendações importantes para facilitar a aplicação da técnica:

- Os Casos de Uso não podem ser nem muito decompostos e nem muito alto nível;
- Pode-se considerar que um cenário de Caso de Uso seja o que equivale a uma transação atômica da proposta de Karner [KARNER,1993];

- Deve-se estabelecer um procedimento para determinar a granularidade dos Casos de Uso e este deve ser seguido coerentemente durante todo o desenvolvimento.

Essas recomendações facilitam o emprego da técnica, pois fornecem importantes diretrizes para o cálculo do esforço. Uma recomendação fundamental do Probasco, foi a de considerar cada transação atômica como um cenário. Não encontramos uma definição exata do que venham a ser transações atômicas, mas o conceito de cenário de Caso de Uso é amplamente difundido como os cursos normais e cursos alternativos que um Caso de Uso pode assumir. Como definido por Larman [LARMAN,2006] um Caso de Uso “Pagamento no Caixa” pode ter os seguintes cenários: “Pagamento em dinheiro”, “Pagamento em Cheque” e “Pagamento com cartão”.

Pelo exposto nesta seção, a técnica Pontos Por Caso de Uso é uma técnica de fácil utilização, porém ela deve ser ajustada à realidade de cada empresa. Tanto pela escolha da granularidade dos Casos de Uso quanto pelo ajuste do nível de esforço NDE calibrado à realidade de cada empresa.

A inexistência de padrões universais para a construção de Casos de Uso dificulta a comparação entre projetos de diferentes organizações [AGUIAR,2006]. Nesses casos, a técnica de Pontos por Função é mais adequada, pois ela é padronizada internacionalmente pela ISO através da Norma ISO/IEC 20926.

3.3 Gerenciamento de riscos

Muitos *gerentes* de projetos de software tomam medidas para assegurar que seus projetos sejam concluídos no tempo previsto e dentro das restrições de esforço e custo [PFLEEGER,2004]. Entretanto, o gerenciamento de projetos envolve muito mais do que simplesmente acompanhar o esforço e o cronograma. Os *gerentes* devem determinar a possibilidade de ocorrência de eventos indesejáveis durante o

desenvolvimento ou a manutenção e fazer planos para evitar esses eventos; ou, caso eles sejam inevitáveis, procurar minimizar as conseqüências negativas. Um risco pode ser considerado como um evento indesejado que tem conseqüências negativas. Os *gerentes* de projeto devem se empenhar no gerenciamento de riscos, a fim de compreendê-los e controlá-los efetivamente em seus projetos.

Muitos eventos ocorrem durante o desenvolvimento de software. Pfleeger [PFLEEGER,2004] distingue eventos de projetos procurando por três aspectos:

- **Uma perda associada ao evento**: o evento deve criar uma situação em que algo negativo acontece no projeto. Uma perda relacionada a tempo, qualidade, dinheiro, controle, compreensão, e assim por diante. A perda associada a um risco é chamada de impacto do risco.
- **A probabilidade de o evento ocorrer**: deve-se ter alguma idéia da probabilidade de o evento ocorrer. A probabilidade do risco é medida de 0 (impossível) até 1 (certeza). Quando essa probabilidade for igual a 1, então o risco será chamado de problema, pois é certo que ele vai acontecer.
- **O grau em que se pode mudar o resultado**: para cada risco, deve-se determinar o que se pode fazer para minimizar ou evitar o impacto do evento. O controle do risco envolve um conjunto de ações tomadas para reduzir ou eliminar um risco. Por exemplo, se os requisitos podem mudar, depois da etapa de projeto, pode-se minimizar o impacto da mudança criando um projeto flexível.

É importante observar que os efeitos dos riscos podem ser quantificados, multiplicando o seu impacto pela probabilidade do risco, para obter a exposição ao risco [PFLEEGER,2004]. Por exemplo, se a probabilidade de que os requisitos se

modifiquem depois de realizado o projeto for de 0,3, e se o custo para refazer o projeto de acordo com os novos requisitos for de R$50 mil, então a exposição ao risco é de R$15mil. Claramente, percebe-se que a probabilidade do risco pode se modificar ao longo do tempo, assim como ocorre com o impacto, de modo que parte do trabalho do *gerente* de projeto é acompanhar esses valores com o decorrer do tempo e planejar as ações de acordo com a exposição ao risco.

Existem duas fontes principais de riscos: os genéricos e os específicos do projeto [PFLEEGER,2004]. Os riscos genéricos são aqueles comuns a todos os projetos de software, tal como a má compreensão dos requisitos, a perda de pessoal fundamental, ou o tempo insuficiente para realizar os testes. Os riscos específicos do projeto são ameaças que resultam das vulnerabilidades particulares de determinado projeto.

Muitos fatores contribuem para mudar as atividades previstas no planejamento. O *gerente* de projeto deve estar atento a todos os fatores que são considerados riscos ao seu planejamento, e tomar ações que visem contornar ou resolver as dificuldades antes que os riscos se tornem problemas reais.

3.4 A Construção de um Plano de Projeto

Para comunicar o cronograma, estimativa de custo do projeto, a descrição técnica do sistema proposto e outros documentos pertinentes ao projeto, pode-se escrever um documento denominado plano de projeto. O plano descreve as necessidades do cliente, assim como o que se espera fazer para satisfazê-las. O cliente pode utilizar o plano para obter informações sobre as atividades do processo de desenvolvimento. Pode-se, também, utilizar o plano para confirmar com o cliente algumas suposições que estão sendo feitas, especialmente sobre o custo e o cronograma.

Um bom plano do projeto inclui os seguintes itens [PFLEEGER,2004]:

1. O escopo do projeto

2. O cronograma do projeto

3. A organização da equipe do projeto

4. A descrição técnica do sistema proposto

5. Os padrões, os procedimentos, as técnicas e as ferramentas propostas para o projeto

6. O plano de garantia da qualidade

7. O plano de gerência de configuração

8. O plano de documentação

9. O plano de gerência de dados

10. O plano de gerência de recursos

11. O plano de testes

12. O plano de treinamento

13. O plano de segurança

14. O plano de gerência de riscos

15. O plano de manutenção

O escopo define os limites do sistema, explicando o que será e o que não será incluído no sistema. Isso assegura ao cliente que foi entendido o que ele quer. O cronograma pode ser expresso utilizando-se uma estrutura de divisão do trabalho, os produtos a serem entregues e uma linha de tempo, a fim de mostrar o que acontecerá em cada momento durante o ciclo de vida do projeto.

Segundo Xavier [XAVIER,2005], para gerar os produtos que o cliente irá receber, no entanto, é preciso gerar outros produtos e serviços no projeto que não estão relacionados no escopo do cliente, como treinamento da equipe do projeto, relatórios de desempenho, contratos, seguros e até mesmo eventos para os quais se costuma dar

pouca importância mas precisam ser executados, como a limpeza de uma sala ou arrumação de cadeiras. Desta forma, o escopo do projeto é bem mais amplo do que o escopo do cliente.

Para representar o escopo do projeto a ferramenta utilizada é a Estrutura Analítica do Projeto (EAP), do inglês *Work Breakdown Structure (WBS)*.

A elaboração da EAP, para Xavier [XAVIER,2005], é geralmente feita decompondo em fases o desenvolvimento do projeto, considerando o "gerenciamento" e o "fechamento do projeto". Identificando os produtos e serviços necessários para propiciar a entrega do escopo do cliente, inclusive os relativos ao gerenciamento do projeto, e verificando as estimativas de custo, tempo, atribuição de responsabilidades e identificação de riscos. Isto tudo para cada *deliverable* (artefato a ser liberado ou produzido ao final de cada fase do projeto) colocado na EAP, de forma a permitir o gerenciamento do projeto (Planejar, executar, controlar e encerrar).

Após sua elaboração, a EAP deve ser ainda revista, refinada e ampliada, até que sejam levadas em consideração todas as necessidades do gerenciamento, e até que o planejamento do projeto possa ser completado.

Os níveis mais Baixos da EAP são denominados de pacotes de trabalho [XAVIER,2005], e formam a base lógica para a identificação de atividades e designação de responsabilidade, estimativa de custos e planejamento de riscos. Deve ser gerado ainda um documento contendo a especificação dos *Deliverables* gerados em cada pacote de trabalho, assim como os critérios de aceitação dos mesmos, formando um "Dicionário da Estrutura Analítica do Projeto" [XAVIER,2005].

Pode-se facilmente perceber que todo o desenvolvimento do projeto fica sob controle constante do "*gerente* do projeto". O plano do projeto também relaciona as pessoas da equipe de desenvolvimento, como elas estão organizadas e o que elas farão.

Nem todo o pessoal é necessário durante todo o tempo de duração do projeto; sendo assim, o plano geralmente contém um diagrama de distribuição de recursos para mostrar a quantidade de pessoal necessária em diferentes momentos.

Muitos documentos são produzidos durante o desenvolvimento, especialmente nos grandes projetos, em que as informações sobre a especificação de projeto devem estar disponíveis para todos os membros da equipe. O plano do projeto relaciona os documentos que serão produzidos, define quem os escreverá e quando isso ocorrerá, e, de acordo com o plano de gerência de configuração, descreve como os documentos serão modificados.

Como todo sistema de software envolve dados para entrada, cálculos e saída, o plano do projeto deve explicar como os dados serão obtidos, armazenados, manipulados e arquivados. O plano também deverá explicar como os recursos serão utilizados.

Para serem efetivos, os testes de software requerem um grande planejamento, e o plano do projeto descreve a abordagem de testes utilizada no projeto. Especificamente, o plano deverá estabelecer como serão gerados os dados de teste, como cada módulo do programa será testado, como os módulos do programa serão integrados e testados, como o sistema inteiro será testado e quem realizará cada tipo de teste.

Cursos de treinamento e documentos geralmente são preparados durante o desenvolvimento, em vez de depois de o sistema estar concluído, de modo que o treinamento possa começar tão logo o sistema esteja pronto. O plano do projeto explica como o treinamento ocorrerá, descrevendo cada aula, software de apoio e documentos, bem como o conhecimento necessário a cada estudante.

Quando um sistema tem requisitos de segurança, é necessário, às vezes,

um plano de segurança separado. O plano de segurança mostra como o sistema protegerá os dados, os usuários e o hardware. Uma vez que a segurança envolve confidencialidade, disponibilidade e integridade, o plano deverá explicar como cada faceta da segurança afeta o desenvolvimento do sistema. Finalmente, se a equipe do projeto será responsável pela manutenção do sistema depois que ele for entregue ao usuário, o plano deverá estabelecer as responsabilidades pelas alterações no código, pelo conserto do hardware e pela atualização da documentação de apoio e dos materiais de treinamento.

3.5 Considerações finais

Este capítulo fez uma breve explicação da métrica APF (Análise por Pontos de Função), e citou todas as etapas do cálculo usadas no método PCU (Pontos por Caso de Uso). Fez também, considerações sobre as adaptações necessárias ao método quando for implantado em algum projeto de software.

Considerou-se a gerência de riscos, pois um planejamento de software, dificilmente ocorre como foi idealizado. Esta seção teve a finalidade de apresentar alguns tipos comuns de riscos e maneiras que eles podem ser diagnosticados e gerenciados.

Finalmente, apresentou-se o conteúdo de um plano de projeto, mostrando os seus componentes e implicações. Tais planos de projeto são requisitos fundamentais nos modelos de qualidade de processo como o MPS.BR [MPSBR,2007].

No próximo capítulo apresentam-se as características dos Métodos Ágeis, pois estes também inspiraram a definição de várias atividades da estratégia proposta neste trabalho. Apresenta-se também o planejamento ágil de sistemas, e uma descrição do método ágil *Scrum* salientando seu processo, suas práticas, papéis e artefatos.

Capítulo 4 – Métodos Ágeis

4.1 Considerações iniciais

Nos últimos anos, Métodos Ágeis de desenvolvimento de software ganharam destaque na indústria de software. Existe uma tendência para o desenvolvimento ágil de aplicações devido ao ritmo acelerado de mudanças na Tecnologia da Informação, pressões por constantes inovações, concorrência acirrada e grande dinamismo no ambiente de negócios [PEREIRA,2007].

Métodos Ágeis apresentam uma abordagem bastante pragmática para o desenvolvimento de software [SATO,2007]. O planejamento detalhado do projeto é realizado para períodos curtos de tempo. O planejamento de longo prazo não é detalhado pois, espera-se que existam adaptações no projeto ao longo do desenvolvimento. Caracteriza-se como sendo uma forma “não tradicional”, como mencionado no Capítulo 3, de desenvolver software, em que a prioridade máxima se concentra na satisfação do cliente através da entrega antecipada e contínua de software susceptível de avaliação [PEREIRA,2007].

Highsmith [HIGHMITH,2002] defende que a indústria e a tecnologia sofrem modificações tão aceleradas que acabam por “atropelar” os métodos tradicionais. Defende ainda, que os clientes são incapazes de definir de forma clara e precisa os requisitos de software, logo no início do projeto em desenvolvimento, o que inviabiliza a adoção de métodos tradicionais em muitos casos.

Considerando que a estratégia proposta neste trabalho tem sido usada nas empresas consideradas nos estudos de caso, em conjunto com o processo do método ágil *Scrum* e que ela mostrou-se bastante adequada a esse método, neste capítulo são apresentadas as características dos Métodos Ágeis com o enfoque no planejamento e no

acompanhamento de projetos. O capítulo está organizado da seguinte forma: na seção 4.2, apresenta-se uma breve caracterização dos Métodos Ágeis; na seção 4.3 são apresentadas as características do planejamento ágil; na seção 4.4 é apresentado o método ágil *Scrum* e na seção 4.5 encontram-se as considerações finais deste capítulo.

4.2 Caracterização dos Métodos Ágeis

A abordagem ágil aplicada à gerência de projetos de software ficou mais clara e melhor definida em 2001, quando um grupo de 17 desenvolvedores experientes, autores e representantes dos mais diversos Métodos Ágeis, se reuniu para discutir um padrão de desenvolvimento de projetos dentre os métodos existentes. Dessa reunião surgiu o Manifesto do Desenvolvimento Ágil [MANIFESTO,2001] de Software que destaca as diferenças com relação aos métodos tradicionais e define seus valores [PEREIRA,2007]:

- **Indivíduos e interações** são mais importantes que processos e ferramentas.
- **Software funcionando** é mais importante que documentação completa e detalhada.
- **Colaboração com o cliente** é mais importante que negociação de contratos.
- **Resposta a mudanças** é mais importante que seguir um plano.

Apesar de reconhecer o valor dos itens à direita, os Métodos Ágeis dão mais importância para os itens destacados à esquerda [PEREIRA,2007]. Este movimento tornou-se a peça-chave do desenvolvimento ágil de software, uma vez que reúne os principais valores dos Métodos Ágeis, que os distingue dos métodos tradicionais de desenvolvimento [DIAS,2005].

Além dos quatro valores básicos, o Manifesto Ágil [MANIFESTO,2001] apresenta doze princípios que diferem o desenvolvimento tradicional do

desenvolvimento baseado em método ágil, a saber:

1. A maior prioridade é satisfazer o cliente através da entrega rápida e contínua de software de valor.
2. Mudanças nos requisitos do sistema são bem vindas, mesmo que de última hora.
3. Entregar versões do software funcionando freqüentemente, em poucas semanas.
4. Cliente e desenvolvedor devem trabalhar juntos ao longo de todo o projeto.
5. Trabalhe com pessoas motivadas, disponibilize o ambiente e o suporte necessário, confie no potencial da equipe.
6. O método mais eficiente para colher informações sobre o projeto é através da conversa "*cara-a-cara*" com o cliente.
7. A principal medida de progresso no projeto é trabalhar sobre software funcionando, e não em documentação.
8. Processos ágeis promovem um desenvolvimento sustentável. Cliente e desenvolvedor devem descobrir o ritmo de trabalho e mantê-lo constantemente.
9. Atenção contínua à excelência técnica e um bom projeto promovem a agilidade.
10. A simplicidade é essencial.
11. As melhores arquiteturas, requisitos e projeto provêm de equipes auto-organizadas
12. Em intervalos regulares, a equipe refletirá sobre como se tornar mais eficiente e, conseqüentemente, poderão ajustar o seu comportamento conforme as necessidades.

Pela compreensão dos princípios dos Métodos Ágeis, pode-se dizer que

eles se caracterizam por serem incrementais, cooperativos, diretos e adaptativos. Incrementais, em função de pequenas versões e ciclos rápidos de desenvolvimento; cooperativos, por estimular a cooperação dos clientes, e a iteração entre os desenvolvedores; diretos, pela simplicidade de aprendizado e documentação reduzida; adaptativos, por aceitar as mudanças ao longo do projeto.

Cockburn [COCKBURN,2002] ressalta que todos os Métodos Ágeis estão baseados em conceitos amplamente conhecidos e adquiridos durante várias décadas de desenvolvimento de engenharia de software, como versões modernas de processos de desenvolvimento iterativos e incrementais, existentes desde a década 70. Apesar de conhecidos desde a década de 70, somente agora a indústria está realmente descobrindo os benefícios dessa abordagem.

Como principais diferenças entre os Métodos Ágeis e os métodos tradicionais podem-se resumir as seguintes:

- Métodos Ágeis são adaptativos e não preditivos: diferentemente dos métodos tradicionais que defendem o planejamento integral do escopo no início do projeto, e um controle rigoroso de mudanças, os planos dos Métodos Ágeis são elaborados e adaptados ao longo do projeto.
- Os Métodos Ágeis são orientados a pessoas e não a processos: nos métodos tradicionais, os processos devem ser projetados para ser independentes das pessoas que os executam. Já os Métodos Ágeis, consideram os indivíduos que irão executar os processos.

A abordagem de simplificação dos Métodos Ágeis, geralmente é confundida com falta de controle e completa anarquia. Na verdade, ter um método simples significa ter agilidade, fazer melhor e, ao contrário do que parece, exige muita disciplina e organização [PEREIRA,2007].

Apesar dos Métodos Ágeis proporem uma mudança do paradigma "comando e controle" para "liderança e colaboração", conforme explica Schwaber [SCHWABER,2004], isto não significa que sejam alheios à necessidade de mensuração do progresso dos projetos. Os Métodos Ágeis podem oferecer um acompanhamento do projeto, por relatório de cada iteração, que contém as datas-chave, comparação de resultados reais versus planejados, métricas e riscos do projeto. Em geral, as práticas de gerenciamento de projeto são aplicadas a cada iteração e não no projeto como um todo.

A adoção de Métodos Ágeis no desenvolvimento de software está vinculada ao contexto no qual o software se aplica. Software cuja precisão e confiabilidade são fatores de decisão e risco (sistema metroviário, aeroviário, cirúrgicos, entre outros) não pode dispensar características dos métodos tradicionais, tais como extensa documentação.

4.3 Planejamento Ágil

Segundo Cohn [COHN,2005], sem agilidade para estimar e planejar não é possível ter projetos ágeis. Portanto, os Métodos Ágeis devem respeitar os valores e os princípios ágeis, retratados no Manifesto para o Desenvolvimento Ágil de Software [MANIFESTO,2001]. O custo com as estimativas e com o planejamento tem grande valor para os autores de Métodos Ágeis. Para Schwaber [SCHWABER,2004], o *Scrum* (que será detalhado na próxima seção) tem um custo menor de planejamento do que um método tradicional, pois no *Scrum* o monitoramento do progresso é feito pelas entregas de software funcionando ao final de cada *Sprint*. Por outro lado, os métodos tradicionais têm um custo alto, pois o desenvolvimento de software é cíclico, e a cada vez que o desenvolvimento volta para uma fase já realizada para fazer um refinamento, por exemplo, o planejamento pode perder a validade e tem que ser refeito.

Para Yoshima [YOSHIMA,2007], muitas vezes não é previsto no

cronograma que, na fase de teste, o processo pode voltar várias vezes para as fases anteriores, como para a codificação, para a análise, ou até para a fase de requisitos. Como é difícil prever quantas vezes esses ciclos irão acontecer, se o planejamento for baseado na dependência entre todas as fases (e quando isso acontece, o gráfico de Gantt é um recurso bastante utilizado), esse planejamento pode ficar defasado em relação ao andamento real do projeto. Yoshima ressalta que as dependências entre as fases do projeto nem sempre se verificam em desenvolvimento de software. Em muitos casos, as fases de análise, codificação e teste ocorrerem quase simultaneamente, ou talvez, em um prazo tão curto que não é interessante gerenciar essas dependências.

O maior problema com o planejamento é que os planos estão quase sempre defasados, segundo Highsmith [HIGHSMITH,2002]. Para ele, deve-se aceitar o fato de que o desenvolvimento de software está envolvido em um ambiente turbulento, com muitas mudanças e, em função disso, os planos devem ser usados como uma guia e não como um mecanismo de controle.

Para Hodgetts [HODGETTS,2004] projetos ágeis devem ser planejados visando à concorrência e o paralelismo das atividades de desenvolvimento. Atividades de todos os tipos podem ocorrer ao mesmo tempo. As oportunidades ditam quando uma atividade deve ser executada e, dessa forma, uma atividade parcialmente feita deve ser considerada como normal. Em gerenciamento de projetos tradicionais, as atividades do mesmo tipo são agrupadas e, preferencialmente, executadas em seqüência e, em última análise, a abordagem cascata é adotada.

Segundo Hodgetts [HODGETTS,2004], os projetos gerenciados de forma tradicional partem do princípio que os processos são definidos, ou seja, está definido o que deve ser feito, quando e como. Para um mesmo conjunto de variáveis de entrada, pode-se esperar o mesmo resultado sempre. Os planos de projeto são criados a partir de

atividades conhecidas, a execução das atividades segue o que foi definido no plano, e o gerenciamento do projeto está em manter a execução das atividades em conformidade com o plano. Porém, segundo Schwaber [SCHWABER,2004] os processos de desenvolvimento de software são empíricos, ou seja, nem todas as características do produto são conhecidas na análise e os requisitos mudam com o tempo. Os processos empíricos devem ser utilizados quando os processos definidos não forem adequados, devido à complexidade do projeto, em que não é possível implantar um processo repetível (que tenha sempre o mesmo resultado, quando mantidas as variáveis de entrada).

É fundamental para o sucesso de um projeto, que todos os participantes tenham a visão de um time unido em torno de um único objetivo comum. É fundamental a visão: “Nós estamos juntos neste projeto”. Segundo Cohn [COHN,2005], analistas não devem passar os requisitos para os projetistas, sem comprometimento. Da mesma forma, todas as interações entre os integrantes do time devem ser realizadas com espírito de equipe e respeito ao trabalho alheio. Para Hodgetts [HODGETTS,2004], deve ser evitado de todas as formas a figura típica do “herói solitário” no desenvolvimento ágil.

Projetos ágeis devem trabalhar com iterações curtas; algumas equipes escolhem trabalhar com iterações de duas semanas, outras conseguem ser ágeis com iterações de até três meses. Para Cohn [COHN,2005], o ideal é escolher o tempo apropriado para a iteração, lembrando que elas devem ter um tempo determinado. Elas devem terminar no tempo estipulado, mesmo que funcionalidades sejam cortadas.

Uma das atividades mais importantes do gerenciamento de projetos ágeis é entregar um software potencialmente pronto para ser usado, ao final de cada iteração. Os cronogramas de trabalho devem ser montados com essa premissa, segundo Hodgetts

[HODGETTS,2004].

O plano criado no início do projeto não tem garantia que irá ocorrer da forma que foi idealizado. Muitos eventos irão conspirar para que os planos sejam mudados – tecnologias podem trabalhar melhor ou pior que esperado, clientes podem mudar suas necessidades, concorrentes podem forçar respostas mais rápidas, etc. Na visão de Cohn [COHN,2004], nestes casos as equipes ágeis devem estar aptas para perceber que mudanças no rumo do projeto são necessárias, e adaptar o projeto para acomodá-las.

Para envolver a equipe de desenvolvimento nas estimativas, é importante que haja o consenso de todos os integrantes do planejamento, sobre as estimativas feitas para as histórias. Diversos autores defendem que esse planejamento deva ser executado com um jogo, em que uma etapa é ganha quando todos os participantes chegam a um consenso sobre a estimativa de uma Estória ou outro produto de trabalho.

Na indústria de software, os Casos de Uso são considerados boa prática para trabalhar com requisitos. Para Hodgetts [HODGETTS,2004], histórias não são Casos de Uso, e nem Caso de Uso são histórias. Casos de Uso definem requisitos funcionais e histórias definem uma parte do sistema que pode ser implementada. O autor propõe que as histórias sejam retiradas dos Casos de Uso.

Nesse contexto, o uso do planejamento e o gerenciamento ágil de projetos têm uma série de benefícios. Cria um ambiente propício para mudança de requisitos e inovações durante os ciclos de desenvolvimento, facilita o gerenciamento de projetos, pois compromete a equipe de desenvolvimento com as decisões, diminui os custos com as mudanças e valoriza a satisfação do cliente com as entregas freqüentes de software.

4.4 Método ágil Scrum

Segundo seu autor, Schwaber [SCHWABER,2004], o *Scrum* não é um processo previsível, pois ele não define o que fazer em todas as circunstâncias. Ele é usado em trabalhos complexos nos quais não é possível prever tudo o que irá ocorrer. Segundo Pereira [PEREIRA,2007], este método oferece um *framework*, que irá servir como uma guia de boas práticas para atingir o sucesso do projeto. No entanto, as decisões de quando e como utilizar tais práticas fica a encargo de quem estiver aplicando.

Para Pereira [PEREIRA,2007], o método *Scrum*, pode ser aplicado em qualquer contexto em que pessoas podem trabalhar juntas para atingir um objetivo em comum. O *Scrum* é recomendado para qualquer área e, principalmente, para projetos de pesquisa e inovação, pois esse tipo de desenvolvimento se baseia em equipes multidisciplinares e auto-organizáveis, nas quais todos os membros têm visão global do objetivo a ser atingido e não existem atribuições fixas, cabendo à equipe se organizar da melhor forma para atingir o objetivo específico.

O *Scrum* implementa um processo iterativo e incremental com três papéis principais: o *Product Owner*, *Team*, e o *Scrum Master* como descrito na Tabela 4.1.

O processo *Scrum* (Figura 4.1) inicia quando o *Product Owner* tem a Visão do produto que irá ser desenvolvido. Segundo Schwaber [SCHWABER,2004], esta Visão contém a descrição geral do que vai ser desenvolvido e o benefício que vai trazer aos usuários deste novo sistema. O *Product Owner* é o responsável por formular e priorizar o *Product Backlog*, que é a lista de requisitos funcionais e não funcionais do novo produto.

Tabela 4.1 - Papéis e responsabilidades do *Scrum*

Papel	Responsabilidade
Scrum Master	Responsável pela gestão do projeto e por liderar as reuniões do *Scrum*. Deve garantir que o *Team* esteja produtivo, livre de interferências externas e de problemas. Facilita a colaboração entre as áreas e as funções. Garante que o processo seja seguido segundo as regras definidas no *Scrum*.
Product Owner	Responsável por definir os requisitos na lista de trabalho (*Product Backlog*). Deve definir as prioridades e data de cada versão do programa (*release*). Deve garantir o retorno financeiro do produto (ROI).
Team	Equipe de desenvolvimento multifuncional e auto-gerenciada. Responsável por selecionar, entre os itens priorizados (*Selected Product Backlog*), os que serão abordados em um *Sprint*. Equipe com liberdade de ação para fazer o que quiser para cumprir o objetivo da iteração.

O trabalho de desenvolvimento é realizado dentro dos *Sprints*. Cada *Sprint* tem um tempo determinado em dias corridos para ser concluído. Schwaber [SCHWABER,2004] propõe que o *Sprint* ocorra em 30 dias.

O planejamento do *Sprint* inicia-se na reunião de "Planejamento 1 do *Sprint"* (Figura 4.1), em que o *Product Owner* apresenta as prioridades do *Product Backlog* para o *Team*. O *Team* deve, então, questionar o propósito e detalhes para obter a compreensão do *Product Backlog*. Após o *Team* obter conhecimento suficiente sobre o *Product Backlog*, ele propõe ao *Product Owner* o "*Selected Product Backlog", que é* uma seleção de requisitos que irá compor uma entrega funcional, de parte do produto, ao final do próximo *Sprint*.

Uma vez estabelecido o "*Selected Product Backlog"* é iniciado o "Planejamento 2 do *Sprint"* (Figura 4.1), no qual o *Team*, que é responsável por gerenciar seu próprio trabalho, deve planejar o *Sprint*. Para tanto, é gerado um conjunto de tarefas que serão executadas nos dias estipulados para o *Sprint*. Esse conjunto de tarefas é conhecido como *Sprint Backlog*.

A cada dia o *Team* realiza uma reunião chamada *Daily Scrum* de 15 minutos. Nessa reunião cada membro do *Team* responde a três questões:

- "O que você fez desde o último *Daily Scrum*?"

-"O que você pretende fazer de agora até o próximo *Daily Scrum*?"

-“Quais foram os impedimentos que atrapalharam seu desempenho?”

O objetivo dessa reunião é sincronizar o trabalho dos membros do *Team*, e agendar uma reunião que o *Team* necessite para dar seqüência no seu trabalho. Com relação à terceira pergunta, que se refere aos impedimentos, pode-se criar uma lista de impedimentos em que o *Scrum Master* deve trabalhar para solucionar. Essa lista é conhecida como “*Impediment Backlog*”.

Ao final de um *Sprint* é realizada a reunião de “Retrospectiva” (Figura 4.1). Nessa reunião o *Team* mostra as novas funcionalidades do produto, que foram desenvolvidas no último *Sprint*, para o *Product Owner,* como também, para qualquer outro interessado no andamento do projeto. Essa reunião deve também ajudar o *Team* a priorizar, de forma colaborativa, o que será feito no próximo “Planejamento 1 do *Sprint”*. Após ter realizado a revisão do *Sprint*, e ter priorizado o trabalho da próxima reunião de planejamento, o *Scrum Master* encoraja o *Team* a revisar o processo de desenvolvimento para torná-lo mais eficiente no próximo *Sprint*.

O *Scrum* é um método ágil fácil de ser aplicado, pois possui poucas práticas, artefatos e papéis. Ele fornece um processo de grande utilidade para o desenvolvimento de software, é capaz de unir o *Team* em torno de um objetivo comum, criar valor para o cliente, e dar condições ao *Scrum Master* de liderar o projeto de forma adaptativa, iterativa e incremental.

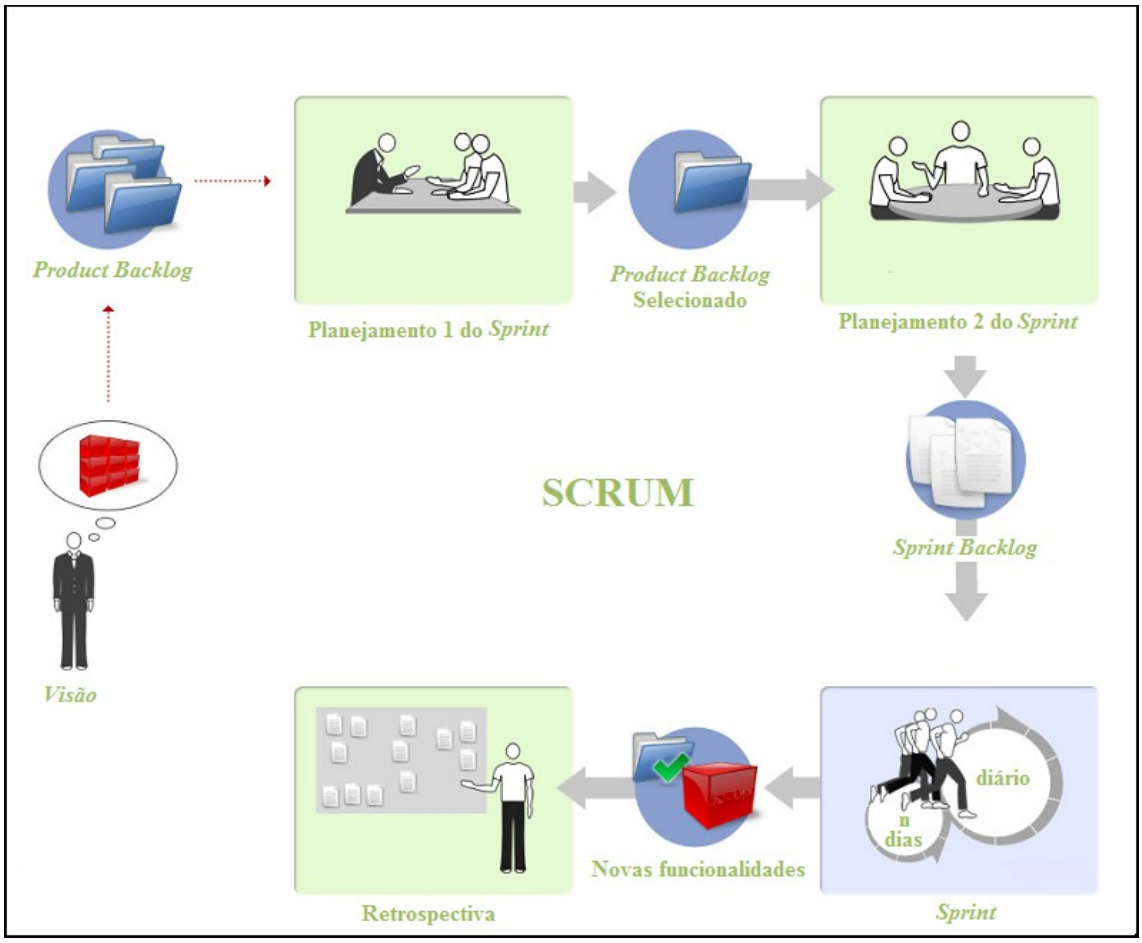


Figura 4.1 - Processo *Scrum* (adaptado de [SCHWABER,2004])

4.5 Considerações Finais

A indústria de software, a cada dia que passa, busca responder mais rapidamente às necessidades do mercado. Nesse contexto, os Métodos Ágeis estão se tornando a chave para alcançar altos níveis de produtividade. De certa forma, a agilidade destes métodos, é uma resposta às falhas de projeto que os métodos tradicionais têm apresentado nessas últimas décadas.

Para efeito de caracterizar esse novo paradigma de desenvolvimento, neste capítulo foram apresentadas as principais características dos Métodos Ágeis, mencionando-se algumas das diferenças entre esses métodos e os métodos tradicionais. Foi dada ênfase ao planejamento e gerenciamento de projetos ágeis, visto que isso tem um relacionamento direto com o trabalho aqui proposto. Por fim, foi apresentado o

método ágil *Scrum*, salientando como suas práticas, papéis e artefatos estão organizados dentro do seu processo, uma vez que a estratégia de planejamento aqui proposta parece ser um recurso bastante apropriado para ser usado no contexto do processo desse método ágil em particular.

No próximo capítulo apresenta-se a Estratégia PCU|$_{PSP}$, que é a proposta deste trabalho, apresenta-se também como o PSP pode ajudar na gerência de projetos, como é a união da Estratégia com o método *Scrum*, e como a Estratégia pode auxiliar na implantação do modelo de qualidade de processo MPS.BR.

Capítulo 5 – PCU|$_{PSP}$: Uma Estratégia de Ajuste de Pontos por Casos de Uso Baseada no PSP

5.1 Considerações iniciais

Neste capítulo, apresenta-se a **Estratégia de Ajuste de Pontos por Caso de Uso Baseada no PSP – PCU|$_{PSP}$** – proposta neste trabalho. Como exposto nos capítulos anteriores, o planejamento é uma área de processo (PA-*Process Area*) fundamental em todos os modelos de qualidade de processo, cuja implantação possui alto grau de complexidade, tanto para grandes empresas quanto para pequenas empresas. Na estratégia aqui descrita, propõe-se uma abordagem que facilita a implantação das atividades de planejamento e controle do processo de desenvolvimento que, embora tenha tido como alvo principal as pequenas empresas, também pode ser aplicada por outras empresas.

Como foi comentado no Capítulo 3, embora a técnica Pontos de Casos de Uso seja uma boa alternativa para dar suporte às estimativas de planejamento de desenvolvimento de software, ela precisa ser ajustada às características do ambiente, principalmente no que diz respeito ao número de horas de trabalho por pontos de casos de uso. Essa é uma informação substancial para que os valores conseguidos da sua aplicação realmente dêem uma boa previsão ao *gerente* do projeto. A estratégia PCU|$_{PSP}$ propõe uma sistemática de uso dos Pontos por Caso de Uso combinada com o processo de melhoria pessoal, o PSP, para que o número de horas de trabalho seja mais preciso e possa ser controlado ao longo do desenvolvimento. Esse controle possibilita um acompanhamento e uma previsão bastante dinâmica de todo o processo, o que permite um ajuste constante do planejamento.

Para apresentar a estratégia proposta, as seções estão organizadas da seguinte forma: na seção 5.2 apresenta-se a Estratégia PCU|$_{PSP}$ descrevendo-se todas as etapas que a compõem, explicando-se como elas devem ser aplicadas e deixando-se os detalhes do uso do PSP no contexto da estratégia para a seção 5.3. Assim, na seção 5.3, apresenta-se uma maneira de aprimorar a equipe de desenvolvimento, por meio do uso do PSP, o que fornece um *feedback* para que o *desenvolvedor* obtenha um maior conhecimento sobre a sua própria capacidade produtiva, como também acrescenta ao *gerente* a função de treinador. Na seção 5.4, relaciona-se a Estratégia PCU|$_{PSP}$ ao método ágil *Scrum*, uma vez que esse método está baseado no aspecto gerencial do desenvolvimento, tendo uma ligação direta com a proposta. Na seção 5.5, discute-se como a estratégia PCU|$_{PSP}$ facilita a implantação do modelo de qualidade brasileiro mais difundido no país, o MPS.BR, analisando-se a vantagem do uso da proposta como facilitador da implantação dos processos definidos nesse modelo.

5.2 A Estratégia PCU|$_{PSP}$

A estratégia PCU|$_{PSP}$ consiste na combinação do modelo de qualidade de processo pessoal PSP – Personal Software Process [HUMPHREY,2000] com o método PCU – Pontos por Caso de Uso [KARNER,1993]. PCU fornece as estimativas de tamanho e de tempo para a codificação do software, enquanto o PSP determina as bases disciplinadas de um processo pessoal de desenvolvimento que visa à melhoria da qualidade e da produtividade. Esta combinação facilita o gerenciamento do desenvolvimento de software e estabelece as diretrizes para que o *gerente* possa exercer, também, a função de treinador. O gerenciamento é facilitado, pois as estimativas são continuamente monitoradas e ajustadas, proporcionando ao *gerente* ferramentas que o ajudam a estimular o *desenvolvedor* a superar suas limitações pessoais.

A estratégia foi estabelecida a partir do uso sistemático e conjunto do

método PCU com o processo PSP, dando suporte à elaboração do planejamento e ao acompanhamento do plano de desenvolvimento, melhorando com isso, a qualidade do processo de software. Na Figura 5.1 apresenta-se um esquema da estratégia PCU|$_{PSP}$.

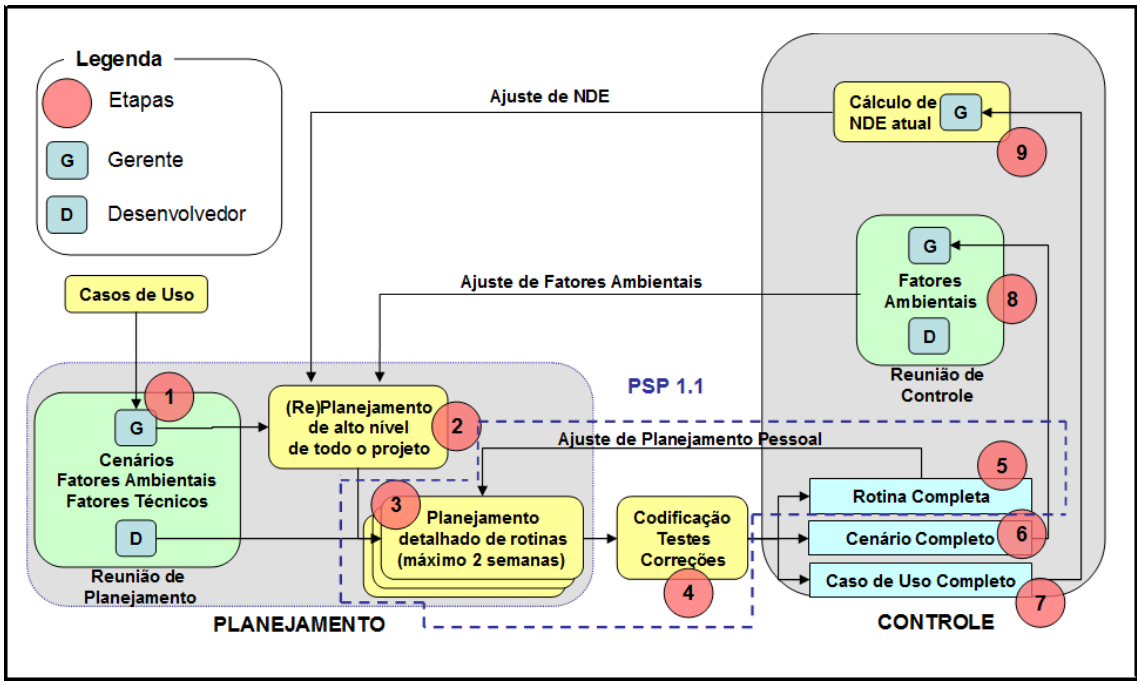


Figura 5.1 - Estratégia PCU|$_{PSP}$

Observe que a estratégia é composta de dois grandes blocos de atividades: um de planejamento e outro de controle que são constantemente executados, complementando um ao outro. O objetivo do bloco de planejamento é permitir que se determinem estimativas adequadas de tamanho, prazos e custos antes do desenvolvimento do sistema ou mesmo de uma rotina, entendida como funções específicas do sistema em questão. O objetivo das atividades de controle é avaliar se o planejamento determinado está sendo obedecido e, em caso negativo, determinar o motivo associado a esse fato. É através do *feedback* destas atividades, que o planejamento é constantemente ajustado.

A estratégia se inicia quando o *gerente* tem disponíveis os Casos de Uso que compõem todo o sistema, ou parte dele (etapa 1). Imediatamente, o *gerente* convoca uma reunião com os Desenvolvedores, a Reunião de Planejamento, na qual são

discutidos todos os cenários de cada Caso de Uso. Essa reunião tem o objetivo de atingir um consenso sobre o entendimento dos cenários, o que permite determinar a complexidade dos Casos de Uso, assim como os níveis de influência dos Fatores de Complexidade Técnica do sistema a ser desenvolvido e dos Fatores Ambientais [KARNER,1993], estes últimos atribuídos para cada *desenvolvedor*.

Com o consenso sobre os casos de uso, o *gerente* pode realizar o planejamento de alto nível de todo o projeto (etapa 2). Nesse momento, através da aplicação do método de Pontos por Caso de Uso ele tem uma estimativa do tamanho do sistema, que é a soma dos Pontos por Caso de Uso, atribuídos a cada *desenvolvedor*. A partir desse valor, para determinar o custo e o tempo de desenvolvimento, o *gerente* deve multiplicar os PCU atribuídos a cada *desenvolvedor* pela variável de Nível De Esforço (NDE), também de cada um deles. O somatório desses valores individuais, fornece a estimativa em horas do tempo de desenvolvimento do sistema. A variável NDE corresponde ao valor de 20 horas homem, proposto por Karner, o qual multiplicado pelos PCU de um sistema, fornece o tempo total de desenvolvimento. Como esse valor varia muito de *desenvolvedor* para *desenvolvedor*, o ideal é que ele seja definido individualmente. É nesse ponto que entra a proposta de uso do PSP, como será visto na próxima seção. Salienta-se que o fato de definir individualmente não significa deixar essa avaliação pública de forma a causar desconforto na equipe. Ao contrário, essa avaliação individual permite que o *gerente* assuma o papel de treinado para ajudar o desenvolvedor a solucionar suas dificuldades.

Na etapa 3, cada *desenvolvedor* utilizará o Planejamento de alto nível da etapa 2 como base para montar o planejamento detalhado de cada cenário. O *desenvolvedor* é responsável por decompor os Casos de Uso em cenários. Cada cenário pode ser decomposto em rotinas menores e ele deve fazer isso da maneira que julgar

mais apropriada. Assim, os Desenvolvedores devem realizar o planejamento de cada uma das rotinas, de acordo com o PSP Nível 1.1, elaborando um cronograma detalhado de desenvolvimento, que inclui também o tamanho das rotinas (em número de linhas de código). Esse cronograma deve estar limitado, no máximo, ao planejamento de duas semanas, mesmo que, para desenvolver todos os cenários que lhe foram atribuídos, o *desenvolvedor* necessite elaborar outros cronogramas, para outros intervalos de duas semanas. Isso evita erros e distorções comuns em longos planejamentos. O cronograma geral do sistema é feito em referência a todos os Casos de Uso e serve como guia mestra para o desenvolvimento de todo o sistema.

O cronograma dos Desenvolvedores deve obedecer às regras de registro de tempo seguindo o processo do PSP, o qual indica que se registre o tempo realmente gasto com o desenvolvimento, separadamente do tempo gasto com as interrupções. Em um dia de oito horas úteis, um *desenvolvedor* pode ter como média cinco horas diárias de programação e três horas são consumidas com outras tarefas adjacentes ao desenvolvimento. Para efeito do cronograma deve-se usar, neste exemplo, cinco horas, pois este é o valor que representa a média de horas realmente utilizadas no desenvolvimento.

O *desenvolvedor* inicia, então, a atividade de desenvolvimento de software (etapa 4), incluindo codificação, testes e correção de defeitos. Ao final do ciclo de desenvolvimento de cada rotina, segue-se uma fase prevista no PSP como *post-mortem* (etapa 5), em que o *desenvolvedor* analisará seu desempenho real em comparação ao desempenho estimado na fase de planejamento (etapa 3). São analisados, portanto, o tempo despendido em relação ao tempo estimado para a rotina desenvolvida, e o tamanho final da rotina em relação ao tamanho estimado em linhas de código. Por meio dessa análise, o *desenvolvedor* deverá ajustar o planejamento de outras

rotinas, caracterizando assim um processo pessoal de melhoria contínua, que o leva a ajustar o seu planejamento para futuro desenvolvimento. Como mencionado anteriormente, os detalhes do uso do PSP, nesta fase da estratégia, serão apresentados na seção 5.3.

Apesar da estratégia aqui proposta incluir somente atividades do nível 1.1 do PSP, o *desenvolvedor* continua com total liberdade para aplicar as demais atividades dos outros níveis; no entanto, essas atividades, por ora, ainda não fazem parte da estratégia aqui proposta, mas serão analisadas em trabalhos futuros para um eventual aproveitamento em outras versões da estratégia.

Quando o *desenvolvedor* conclui um conjunto de rotinas que caracteriza completamente um cenário (etapa 6), o *gerente* deve ser informado do fato. Este, por sua vez, compara o desempenho real do *desenvolvedor* em relação ao planejamento de alto nível, desenvolvido na etapa 2. Se houver discrepância significativa, o planejamento de alto nível do sistema pode estar ameaçado e, portanto, é necessária uma reunião de Controle, entre *gerente* e *desenvolvedor*, para identificar as causas de tal discrepância e buscar, em conjunto, meios de transpor tais dificuldades (etapa 8).

É muito importante que o *gerente* tenha uma postura pró-ativa em relação ao incentivo de melhoria do *desenvolvedor*, uma vez que, na reunião de planejamento, buscou-se o consenso sobre a complexidade do Caso de Uso. O erro, nesse caso, pode estar no planejamento de ambos os lados. Assim, realiza-se a reunião de Controle entre *gerente* e *desenvolvedor*, cujo objetivo é encontrar, de comum acordo, um meio para contornar os problemas que ocorreram. Exemplificando, o *desenvolvedor* pode informar ao *gerente* que, ao contrário de sua opinião inicial, ele não possui domínio suficiente de uma linguagem de programação e, portanto, solicita um tempo maior de treinamento na linguagem. O *gerente*, por sua vez, atualiza o nível de influência do Fator Ambiental

“Dificuldade com a Linguagem de Programação”, aumentando o seu valor. Tal aumento tem impacto na técnica de Pontos por Caso de Uso que, por sua vez, tem impacto no Planejamento de Alto Nível do projeto (etapa 2).

Na metodologia proposta por Karner [KARNER,1993], os Fatores Ambientais são analisados para a equipe de desenvolvimento como um todo. Mas em um ambiente de desenvolvimento de uma pequena empresa, esta estratégia propõe que os Fatores Ambientais sejam analisados individualmente, para cada *desenvolvedor*.

Ressalta-se então, que no acompanhamento realizado entre as etapas 6, 8 e 2, apenas o nível de influência dos Fatores Ambientais necessita ser alterado, uma vez que os Fatores Técnicos se mantêm inalterados quando os requisitos do sistema também assim se mantêm. Essa atividade de ajuste do planejamento de alto nível se caracteriza também como uma melhoria contínua do processo de desenvolvimento como um todo. A cada cenário concluído, repete-se esse ciclo e o planejamento de alto nível, caso seja impactado, é novamente ajustado.

Os Fatores Ambientais interferem significativamente no cálculo dos Pontos por Caso de Uso e fornecem indícios sobre o desempenho do *desenvolvedor*. Na estratégia $PCU|_{PSP}$ os Fatores Ambientais devem ser avaliados individualmente, pois o foco aqui são pequenas equipes de desenvolvimento de software e o controle individual pode melhorar os resultados finais. Como a influência de cada fator varia entre 0 a 5, como foi visto no capítulo 3, se considerarmos a situação apresentada na Tabela 5.1 para o melhor *desenvolvedor* possível, o cálculo do Fator Ambiental resulta em 0,64 (FA=0,64), e para o pior *desenvolvedor* possível, resulta em um Fator Ambiental de 1,46 (FA=1,46), com uma variação de 228,13 % de esforço do melhor para o pior. Dessa forma, pode-se perceber que a experiência, motivação e a formação de um *desenvolvedor* influenciam, significativamente, no esforço em horas, na construção de

um sistema.

Tabela 5.1 - Fatores ambientais

Fator	Descrição	Peso (E*i*)	Influência Melhor	Influência Pior
E1	Familiaridade com o processo de desenvolvimento.	1,5	**5**	**0**
E2	*Desenvolvedores em meio expediente.*	*-1*	***3***	***3***
E3	Analista experiente	0,5	**5**	**0**
E4	Experiência com a aplicação em desenvolvimento.	0,5	**5**	**0**
E5	Experiência em Orientação a Objetos.	1	**5**	**0**
E6	Motivação	1	**5**	**0**
E7	Dificuldade com a linguagem de programação	-1	**0**	**5**
E8	*Requisitos estáveis*	*2*	***3***	***3***

Com relação aos Fatores de Complexidade Técnica obeserva-se que: como o desenvolvimento de software nem sempre é um processo totalmente definido, em muitos casos, a implementação de um requisito funcional ou mesmo uma mudança nos requisitos pode levar à escolha de uma tecnologia diferente da planejada inicialmente. Essas mudanças podem impactar diretamente nos Fatores de Complexidade Técnica que, analogamente ao ajuste dos Fatores Ambientais, interferem no planejamento do software como um todo. Portanto, o *gerente* deve estar atento às mudanças que interferem nos Fatores de Complexidade Técnica e a influência dos mesmos no controle do desenvolvimento do software. Por exemplo, quando um cliente solicita a portabilidade entre os sistemas operacionais Linux e Windows, não requerida anteriormente, o *gerente* deve refazer o planejamento de alto nível levando em consideração nos cálculos das estimativas, a mudança do Fator de Complexidade Técnica.

Em função da aplicação do PSP, o *gerente* deve assumir o papel de um treinador para incentivar o *desenvolvedor* a vencer suas limitações pessoais de

planejamento, produtividade e qualidade. A perspectiva de um *gerente* "treinador" é fundamental para a gerência de projetos nesta estratégia, pois o clima ideal deste processo de desenvolvimento é de auto superação dos Desenvolvedores. A todo o momento, o *desenvolvedor* deve procurar melhorar suas "marcas" assim como um atleta sempre procura superar seus recordes. O *gerente* deve assumir o papel de treiandor para trabalhar as dificuldades de cada *desenvolvedor* individualmente. O trabalho de treinador consiste em ajudar os Desenvolvedores a vencer as dificuldades, tomando como base os dados fornecidos pelo uso do PSP, analisados à luz da experiência acumulada do *gerente*.

Quando todos os cenários de um Caso de Uso foram desenvolvidos (etapa 7), o *gerente* deve comparar o esforço estimado para o desenvolvimento desse Caso de Uso com o tempo real gasto para tal atividade. Assim, é possível ajustar o NDE (etapa 9) de cada *desenvolvedor* logo no primeiro Caso de Uso concluído, para que este reflita a verdadeira relação entre Pontos por Casos de Uso e o esforço em horas/homens, para cada *desenvolvedor* em particular.

O mecanismo de ajuste é bem simples, bastando dividir o tempo real gasto, fornecido pelo PSP após o Caso de Uso ser concluído, pela soma dos Pontos dos Casos de Uso já desenvolvidos pelo *desenvolvedor*. Tal mecanismo deve ser repetido sempre que um Caso de Uso for finalizado (etapa 7), ou seja, acumulam-se o tempo gasto e os Pontos por Caso de Uso já realizados e divide-se um pelo outro. Por exemplo, na Tabela 5.2, apresentam-se cinco Casos de Uso de um sistema hipotético, considerando-se o uso da $PCU|_{PSP}$. As colunas da tabela apresentam respectivamente: Caso de Uso, Complexidade do Caso de Uso, Pontos por Caso de Uso Ajustados (individual e acumulado), Horas efetivamente consumidas para conclusão do Caso de Uso (individual e acumulada), o NDE que é calculado dividindo-se as horas acumuladas

pelos Pontos por Casos de Uso acumulados, Total de Horas estimadas para desenvolvimento do sistema, com base no valor do NDE, recalculado a cada Caso de Uso. Para este exemplo estipulou-se os seguintes valores: dois atores complexos (6 pontos), Fator Ambiental em 0,73 (FA=0,73), e o Fator de Complexidade Técnica em 1,12 (FCT=1,12). Ressalta-se que os pontos referente à complexidade dos atores (6 pontos não ajustados) foram ajustados aos Fatores Ambientais e Técnicos, e distribuídos de forma igualitária entre todos os casos de uso.

Tomando-se como exemplo o Caso de Uso 3, o NDE do *desenvolvedor*, neste momento, é calculado dividindo-se as horas acumuladas (110 horas) pelos Pontos por Caso de Uso acumulados (27,16 pontos) obtendo-se o valor 4,05. Com esse nível de esforço, calcula-se novamente a quantidade de horas para o desenvolvimento do sistema como um todo (199,70 horas), multiplicando-se o novo NDE pelo total de Pontos por Casos de Uso do sistema (49,31 pontos). Isso significa que se o desenvolvimento transcorrer dessa forma, desse ponto em diante, o desenvolvimento do sistema seria concluído em 199,70 horas.

Tabela 5.2 - Exemplo Hipotético da Aplicação da Estratégia PCU|$_{PSP}$ (Ver Figura 5.1)

	Etapa 1	**Etapas 1 e 2**		**Etapa 4**		**Etapa 9**	**Etapa 2**
Casos de Uso (UC)	**Complexidade do Caso de Uso**	**Pontos por Caso de Uso ajustados**		**Horas Efetivamente Consumidas para conclusão do Caso de Uso**		**NDE**	**Horas Previstas**
		Individual	**Acumulado**	**Individual**	**Acumulado**		
UC 01	Complexo	13,10	13,10	48	48	3,67	180,74
UC 02	Simples	5,01	18,11	22	70	3,87	190,63
UC 03	Médio	9,05	27,16	40	110	4.05	199,70
UC 04	Complexo	13,10	40,26	70	180	4,47	220,48
UC 05	Médio	9,05	49,31	23,5	203,5	4,13	203,50

No primeiro Caso de Uso, o *gerente* estima que o sistema pode ser desenvolvido em 180,74 horas. Porém, conforme a tabela acima, verifica-se que foi concluído em 203,5 horas. O *gerente* deve, então, estar atento às variações do NDE, pois se seus valores variarem consideravelmente, ele pode avaliar, imediatamente, o

impacto em seu planejamento inicial. O *gerente* deve avaliar se as causas de variações significativas são ocasionais ou se representam uma tendência. Quando se configura uma tendência, o impacto no cronograma pode ser grande e, em muitos casos, o *gerente* deve negociar com os interessados uma alteração no cronograma ou então, encontrar maneiras para que os prazos fiquem compatíveis com as necessidades dos clientes. A estratégia PCU|$_{PSP}$ permite ao *gerente* ter indícios do andamento do desenvolvimento do sistema, acompanhando as variações do NDE, bem como tomar ações corretivas no início de um eventual atraso no cronograma.

A Estratégia PCU|$_{PSP}$ pode ser aplicada a uma equipe com vários Desenvolvedores. Nesse caso, o *gerente* deve montar o cronograma geral do projeto aplicando a Estratégia PCU|$_{PSP}$ para cada *desenvolvedor*. A Figura 5.2, ilustra a aplicação da Estratégia, em um sistema hipotético, em que quatro Desenvolvedores fazem parte da equipe. Na barra horizontal que representa a alocação do *desenvolvedor* no projeto, ressaltou-se que existe um NDE e um FA (Fator Ambiental) para cada *desenvolvedor*.

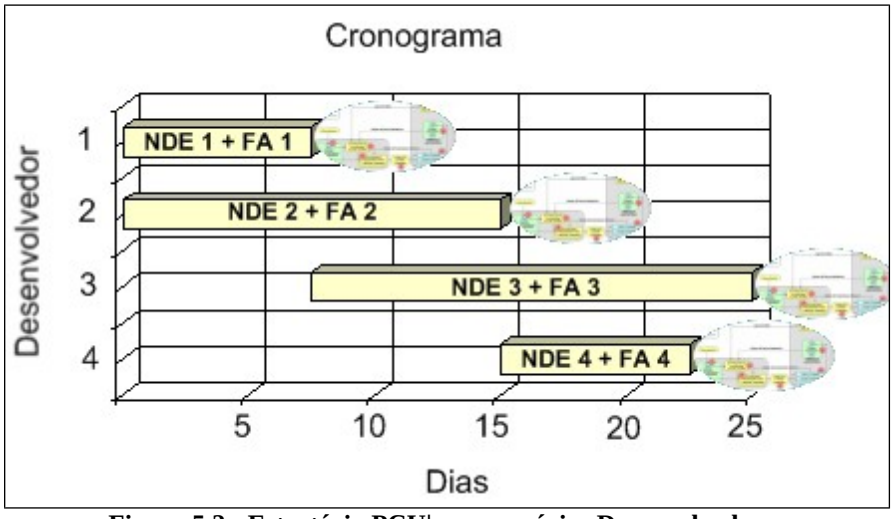


Figura 5.2 - Estratégia PCU|$_{PSP}$ com vários Desenvolvedores

Com o estudo do método PCU e do processo PSP, constatou-se que, separadamente, o método PCU abrange apenas o planejamento do software sem

oferecer mecanismos de controle do desenvolvimento, enquanto que o processo PSP se restringe à melhoria pessoal do *desenvolvedor*, não sendo suficiente para o desenvolvimento do software como um todo. Unindo o método PCU ao processo PSP, definiu-se uma estratégia de planejamento e controle do desenvolvimento de um projeto de software bem como o aprimoramento e o controle da equipe de Desenvolvedores, propiciando uma melhoria na produtividade e, conseqüentemente, um melhor desempenho de toda equipe no decorrer do projeto como um todo.

5.3 Estratégia PCU|$_{PSP}$ e o Processo PSP

A Estratégia PCU|$_{PSP}$ tem seu alicerce no processo PSP, desenvolvido por Humphrey [HUMPHREY,2000]. Conforme explicado anteriormente, este processo estabelece meios pelos quais os Desenvolvedores conseguem melhorar sua produtividade, diminuir sua taxa de erros e melhorar continuamente a capacidade de fazer previsões tanto com relação ao tamanho do software a ser desenvolvido em Linhas de Código (LOC – *Lines of Code*), quanto com relação ao tempo de desenvolvimento.

O *gerente* utiliza algumas informações geradas através do emprego de atividades do PSP para avaliar o desempenho dos Desenvolvedores e ajustar o planejamento do software. As informações que o *gerente* utiliza são derivadas do uso PSP, até o nível 1.1, assinaladas na Figura 5.3.

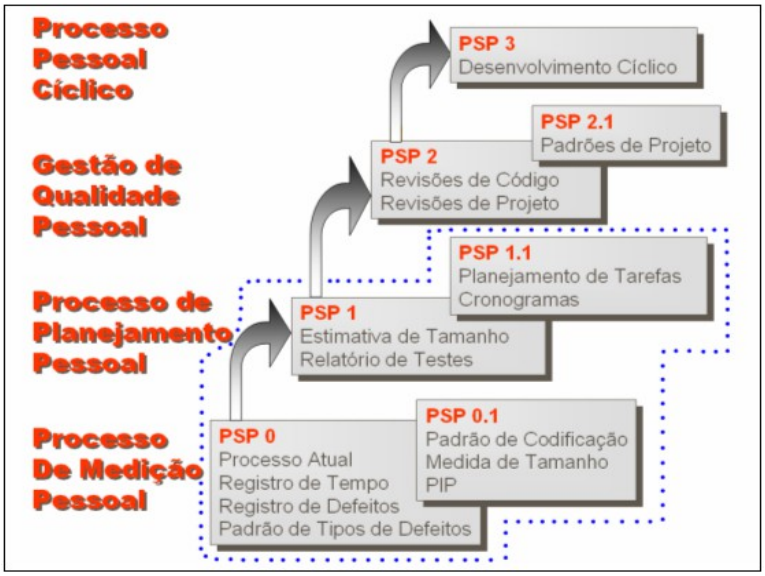


Figura 5.3 - Níveis do PSP (Adaptado de [HUMPHREY, 2000])

O *gerente* deve assumir uma postura de treinador do *desenvolvedor*, incentivando-o a executar as atividades propostas pelo PSP como meio de auto-aperfeiçoamento, muito mais como um facilitador, do que como um controlador. Essa postura objetiva o comprometimento do *desenvolvedor* com o seu próprio processo de melhoria. Ele deve ter em mente o fato de que não se trata de uma imposição, mas de um recurso oferecido pelo *gerente*, que propicia a melhoria contínua de sua produtividade e, conseqüentemente, o aprimoramento do processo pessoal de desenvolvimento de software.

É indicado ao *gerente* seguir a implantação do PSP gradualmente, de acordo com os níveis estabelecidos nesse processo. Dessa forma, o primeiro passo é a implantação do Processo de Medição Pessoal (Figura 5.3), que comporta os níveis PSP 0 e PSP 0.1. É justamente nesse passo que o *desenvolvedor* adquire o hábito de registrar os tempos e inicia um processo denominado de auto-conhecimento.

Na fase PSP 0, são executados o “Processo Atual” que se desenvolve através de seis fases padrão (Análise, Projeto, Codificação, Compilação, Testes e *Postmortem*), “Registro de Tempos” (levando em consideração as interrupções), “Registro de Defeitos” e “Padrão de Tipos de Defeitos”. O *gerente* deve respeitar o

tempo de aprendizado do *desenvolvedor* e mudar de nível quando o PSP 0 for completamente compreendido.

O nível do PSP 0.1 (Figura 5.3) acrescenta ao *desenvolvedor* as atividades de “Padrão de Codificação”, “Medida de Tamanho” (em linhas de código) e propostas de melhoria do Processo “PIP” (*Process Improvement Proposals*). O PIP é um formulário, por meio do qual o *desenvolvedor*, após analisar seu desempenho, faz uma proposta de melhoria de seu processo pessoal. Essa proposta deve ser concretizada no próximo ciclo de desenvolvimento, e o *desenvolvedor* deve estabelecê-la como meta. A cada novo ciclo, ele deve identificar o que funcionou e o que não funcionou na proposta, modificando-a, a fim de fortalecer os acertos e evitar os erros. O *gerente* deve ajudar o *desenvolvedor* a identificar as suas dificuldades e, principalmente, a encorajá-lo a superar seus próprios obstáculos profissionais.

Uma vez incorporados os níveis PSP 0 e PSP 0.1, o *gerente* deve continuar a motivar a utilização dos níveis seguintes do PSP, para incorporar o planejamento pessoal de cada *desenvolvedor*. No nível PSP 1.0, o *desenvolvedor* inicia as atividades de “Estimativa de Tamanho” (na Estratégia $PCU|_{PSP,}$ refere-se ao planejamento das rotinas dos Casos de Uso, realizadas na Etapa 3).

Na Figura 5.4 (gráfico gerado na ferramenta Process Dashboard, descrita no Capítulo 2), observa-se o gráfico de erro na estimativa de tamanho realizada pelo *desenvolvedor*, na etapa 3 da Estratégia $PCU|_{PSP,}$ relativa às rotinas que compõem os Casos de Uso atribuídos ao *desenvolvedor* quando do desenvolvimento de um determinado sistema. Observa-se, no eixo horizontal, as rotinas feitas pelo *desenvolvedor* e, no eixo vertical, o erro cometido por ele, na estimativa do tamanho de cada rotina, em LOC.

Com a análise do gráfico, percebe-se que o *desenvolvedor* em questão

mantém uma regularidade nos acertos do tamanho das rotinas, porém, em algumas delas o erro atinge picos de até 1000%. O *gerente* deve estar atento a esses gráficos e, juntamente com o *desenvolvedor*, tentar compreender o motivo pelo qual houve um grande erro de estimativa, auxiliando-o a acertar mais na próxima tentativa.

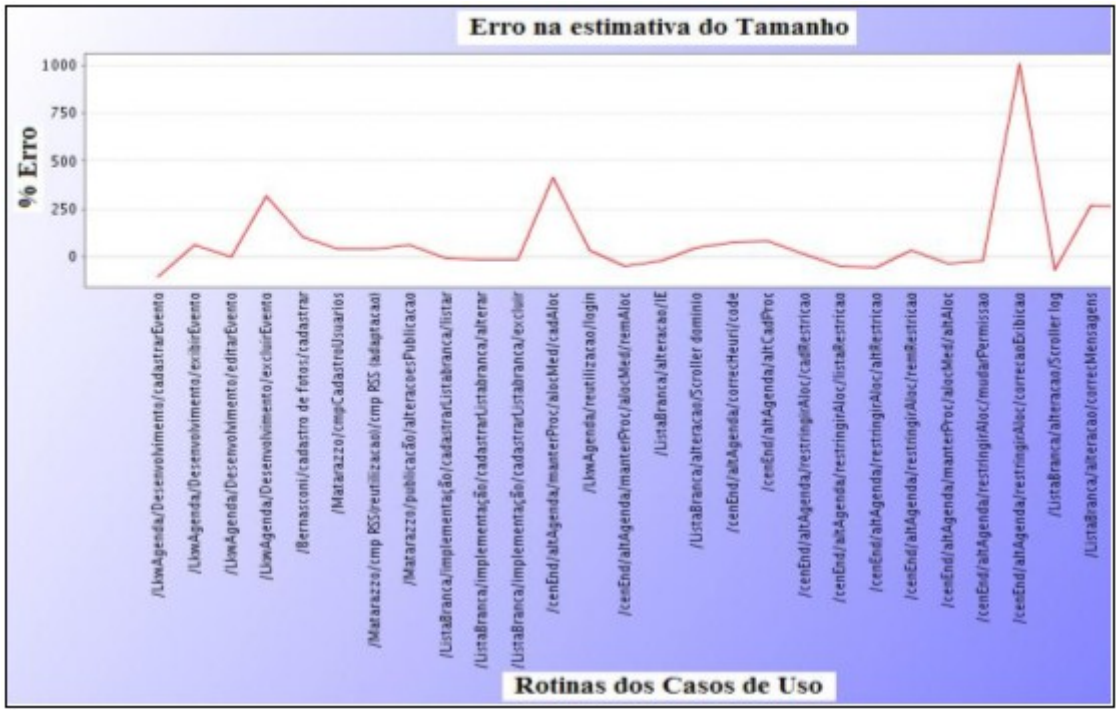


Figura 5.4 - Exemplo de erro na estimativa do tamanho de rotinas.

As Figuras 5.4, 5.5 e 5.6, cujos gráficos foram gerados com a ferramenta Process Dashboard [DASHBOARD,2007], fazem parte do estudo de caso apresentado no Capítulo 6, mas estão apresentadas também neste capítulo para que se possa explicar como o PSP é usado na estratégia proposta neste trabalho.

A Estratégia $PCU|_{PSP}$ tem o foco principal no planejamento e no acompanhamento do desenvolvimento do software. Portanto, o *gerente* deve incentivar os Desenvolvedores a elaborarem os formulários PIP, no sentido de melhorar seu processo pessoal de planejamento.

Como o treinador de um atleta, o *gerente* deve incentivar os Desenvolvedores a identificarem suas dificuldades e deficiências, para que estas possam

ser tratadas de forma apropriada, o que é feito, através dos formulários PIP, nas Reuniões de Controle da Estratégia PCU|$_{PSP.}$ Portanto, o objetivo principal do *gerente* é ajudar os Desenvolvedores a identificar e avaliar suas dificuldades, buscando perceber, com sua experiência, todos os fatores que podem impedi-los de atingir suas metas de melhoria do processo pessoal. O *gerente* ainda deve apoiá-los, de modo que possam superar suas dificuldades profissionais. Dessa forma, o planejamento será melhorado e as atividades de acompanhamento do desenvolvimento de software serão realizadas de forma mais efetiva.

No nível PSP 1.1 (Figura 5.3), o *desenvolvedor* deve fazer o "Planejamento das Tarefas" e o "Cronograma". Na Estratégia PCU|$_{PSP,}$ esta atividade está representada pela etapa 3 – (Figura 5.1). Nessa fase, o *desenvolvedor* faz estimativas do tempo para o desenvolvimento de cada rotina que compõe o cenário de um Caso de Uso, e planeja as tarefas que irá executar. Baseado no planejamento de cada rotina, o *desenvolvedor* monta o seu cronograma, momento em que este deve ser aconselhado a não ultrapassar duas semanas de cronograma detalhado. O *gerente*, então, deve avaliar se o cronograma e o tempo que o *desenvolvedor* alocou estão condizentes com a realidade, baseando-se nos dados históricos do *desenvolvedor*, provenientes do registro de tempo fornecido pelo uso sistemático do processo PSP.

A Figura 5.5 mostra um exemplo de erro na estimativa de tempo de um *desenvolvedor*. Por meio do gráfico da Figura 5.5, observa-se que nas primeiras quatorze rotinas, houve uma constância do *desenvolvedor* no cálculo da estimativa de tempo; porém, nas três rotinas seguintes ocorreu uma inconstância, que pode prejudicar o planejamento do Software como um todo.

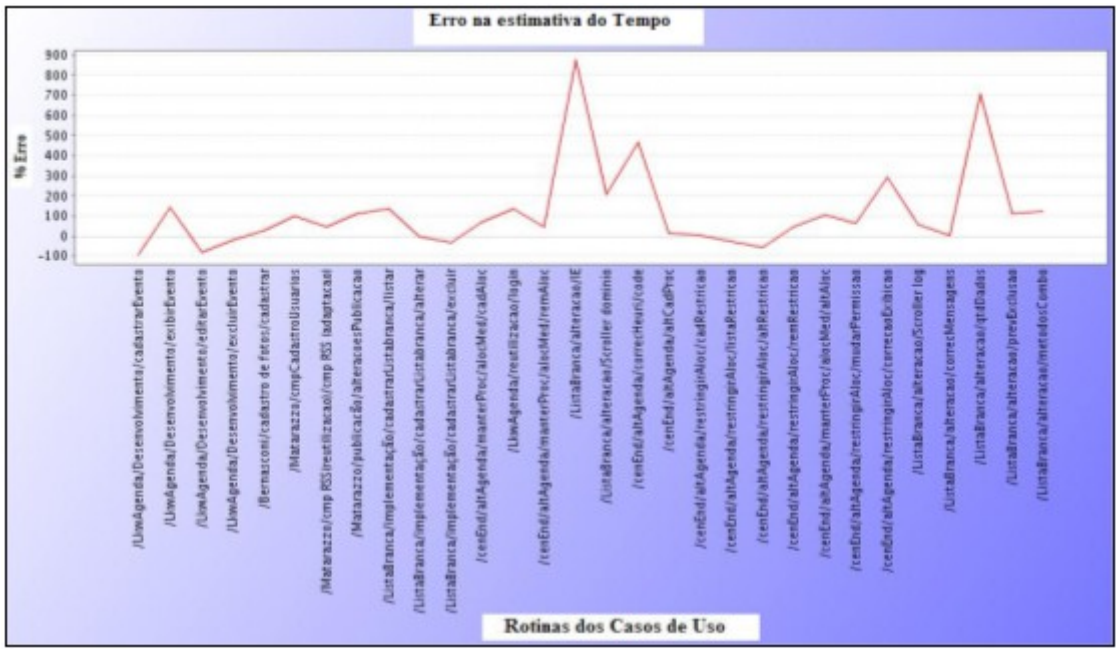


Figura 5.5 - Exemplo de Erro na estimativa do tempo.

Através do conhecimento dessas irregularidades no gráfico, o *gerente* pode indicar treinamentos, de forma que o *desenvolvedor* adquira maior conhecimento de toda a tecnologia utilizada e, com isso, seja capaz de produzir software com maior qualidade. Certamente, resolvendo essas dificuldades, além de produzir um bom software, isso terá impacto direto no seu desempenho. Para o *desenvolvedor*, o conhecimento de suas próprias inconstâncias proporciona um maior controle sobre sua produtividade e permite que ele previna possíveis problemas de execução de planejamento de suas atividades.

Portanto, verifica-se o quanto o planejamento pessoal de cada *desenvolvedor*, postulado pelo processo PSP, estrutura o planejamento final das tarefas realizadas pelo *desenvolvedor* na Estratégia $PCU|_{PSP}$.

A Figura 5.6 exibe o tempo gasto pelo *desenvolvedor* na atividade de planejamento para cada rotina considerada na Figura 5.4. Os detalhes de relacionamento entre os gráficos mostrados serão discutidos no Capítulo 6. Entretanto, é importante observar que todas essas estatísticas compreendem aspectos ortogonais do

desenvolvimento e auxiliam definitivamente o *gerente* a discutir as alternativas de melhoria com seus Desenvolvedores.

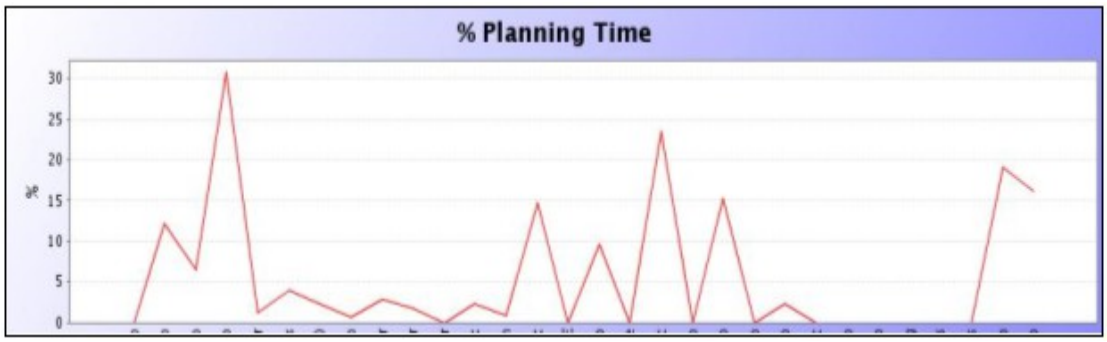


Figura 5.6 - Tempo de planejamento gasto por um *desenvolvedor*

5.4 Estratégia PCU|$_{PSP}$ e o método ágil Scrum

A Estratégia PCU|$_{PSP}$ se adequa ao método ágil *Scrum*, estudado no Capítulo 4 deste trabalho, de forma bastante natural. Um dos princípios do método *Scrum*, conforme já mencionado, está relacionado às atividades de inspeção e adaptação, desde que o método assuma que nem todas as características do produto são conhecidas na atividade de análise. A proposta do *Scrum* contempla uma visão empírica baseada em Métodos Ágeis, característica que torna este método ideal para a aplicação da estratégia.

É importante ressaltar que a Estratégia PCU|$_{PSP}$ e o método *Scrum* formam um conjunto harmônico, já que um complementa o outro. Se por um lado, o processo do *Scrum* define apenas “o que deve ser feito” [KNIBERG,2007], por outro, a estratégia PCU|$_{PSP}$ trata especificamente de “como deve ser feito”. Portanto, algumas questões que não são respondidas pelo método *Scrum*, são abordadas pela estratégia PCU|$_{PSP}$. São elas:

1) Como estimar o *Product Backlog*?

2) Como inserir itens no *Selected Product Backlog*?

3) Como estimar as tarefas do *Sprint Backlog*?

4) Como gerenciar o *Burndown Chart* e a velocidade da equipe de

desenvolvimento?

5) Como o *Scrum Master*, na qualidade de treinador, avalia a evolução dos Desenvolvedores?

Quanto aos papéis que fazem parte do *Scrum*, pode-se identificar um mapeamento bastante natural entre os papéis da estratégia PCU|PSP e os do método *Scrum*, satisfazendo as suas principais características. O *Scrum Master* é representado pelo *gerente* e a equipe de *desenvolvedores* correspondem ao *Team*. O *Product Owner* é o representante do cliente, para a equipe de *desenvolvedores*, sendo responsável pelo retorno do investimento.

O conceito de *Scrum Master* tem a mesma essência do conceito de *gerente* descrito na Estratégia PCU|PSP. Em ambos esse papel deve assumir uma postura de treinador ao invés da postura assumida por um chefe tradicional. Esse papel deve servir como um facilitador, ao invés de um controlador [SCHWBER,2004], pois com essa postura, se obtém um maior comprometimento da equipe de *desevolvedores* com a entrega do produto final, uma vez que, em ambos os casos, há um mesmo objetivo: a melhoria na produtividade dos Desenvolvedores.

Com relação ao processo estabelecido pelo *Scrum*, a Estratégia proposta se adapta a esse método contemplando as seguintes atividades: Planejamento 1 do *Sprint*, Planejamento 2 do *Sprint*, atividades de acompanhamento do *Sprint* e a atividade de Retrospectiva.

Na Figura 5.7, apresenta-se o processo do *Scrum* identificando-se as etapas da Estratégia PCU|PSP que podem apoiar e sistematizar as fases desse processo, servindo como um complemento a ele.

De acordo com a estratégia PCU|PSP, pressupõe-se que o repositório de requisitos levantados em fases anteriores do processo, que no método *Scrum* é

denominado *Product Backlog* esteja organizado e bem definido.

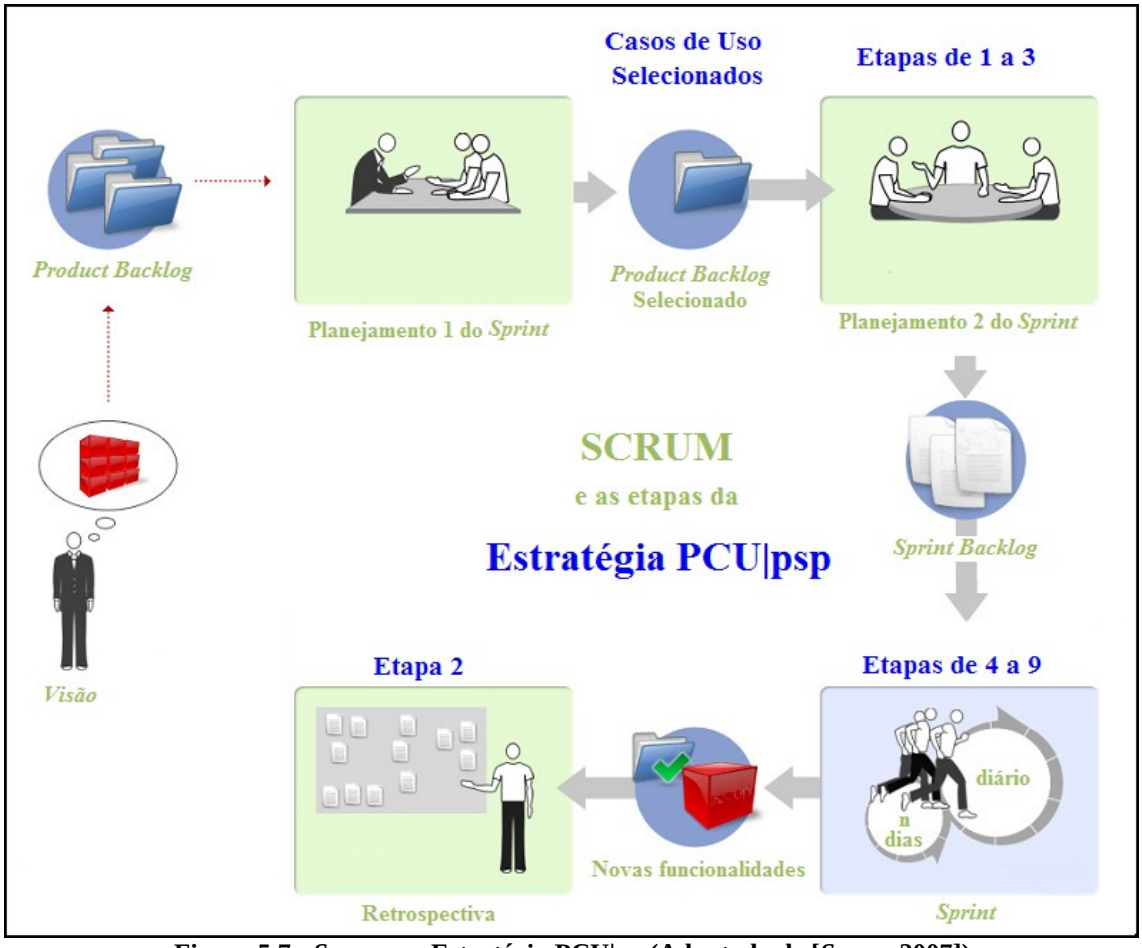


Figura 5.7 - *Scrum* e a Estratégia PCU|$_{PSP}$ (Adaptado de [*Scrum*,2007])

A partir desse repositório, inicia-se a atividade denominada Planejamento 1 do *Sprint*, na qual o *Product Owner* se reúne com o *Scrum Master* e o *Team* para definir as prioridades de desenvolvimento do produto. O papel de *Product Owner* e a atividade Planejamento 1 do *Sprint* não foram previstos na Estratégia PCU|$_{PSP}$. Porém, essa Estratégia contribui para a atividade Planejamento 1 do *Sprint*, na seleção dos itens que compõem o *Product Backlog*, o qual é definido através das estimativas de tamanho e tempo da Estratégia PCU|$_{PSP}$. As estimativas provêm da atividade de elaboração e refinamento de Casos de Uso (Planejamento 1 do *Sprint*), decompondo-os em cenários, de forma a subsidiar a contagem de pontos.

Para cada Caso de Uso, é determinada a sua complexidade, de acordo com os mecanismos de cálculo estabelecidos pela estratégia PCU|$_{PSP,}$ explicados na

seção 5.2. Diferentemente do método do PCU definido por Karner [KARNER,1993], a estratégia proposta permite calcular o tempo despendido para desenvolver cada Caso de Uso, baseando-se nos pontos e no NDE do *desenvolvedor*. Essa abordagem traz enormes benefícios à atividade de seleção dos itens do *Product Backlog* na medida que facilita a composição do *Product Backlog*, o qual é definido respeitando-se o tempo determinado para o *Sprint*.

O *Product Backlog* corresponde à seleção de Casos de Uso, utilizada na etapa 1 da Estratégia PCU|$_{PSP}$ (Figura 5.1). Com o *Product Backlog* definido, executa-se a atividade Planejamento 2 do *Sprint* que se relaciona às etapas 1, 2 e 3, referentes à etapa de Planejamento da Estratégia PCU|$_{PSP}$, e tem o propósito de definir o prazo e as tarefas que serão executadas no *Sprint* seguinte. Primeiramente, realiza-se a revisão da complexidade dos Casos de Uso, com todos os membros do *Team*. Após isso, decompõem-se os cenários em rotinas mais granulares, realiza-se a estimativa de cada rotina e o agendamento pessoal do *desenvolvedor*. Entende-se por decomposição de cenários em rotinas mais granulares por uma atividade proposta pela estratégia PCU|$_{PSP,}$ que tem por objetivo dividir cada cenário em uma seqüência finita de tarefas inter-relacionadas que, quando executadas conjuntamente, produzem o cenário correspondente.

Como foi detalhado na seção 5.2, recomenda-se que o planejamento detalhado não deve ser longo e também, para efeito de agendamento diário, as horas alocadas por dia deve corresponder à carga horária destinada às atividades do projeto, descontando-se as horas consumidas em outras atividades. Estas recomendações respeitam os princípios usados em planejamento dos Métodos Ágeis.

Após a atividade de decomposição, inicia-se a estimativa de cada rotina, tanto em relação ao tempo de desenvolvimento quanto em relação à quantidade de

linhas de código (LOC). Nesse momento, as atividades do PSP devem ser aplicadas para que o *desenvolvedor* melhore a taxa de acertos de suas próprias estimativas. A cada ciclo de desenvolvimento, o *desenvolvedor* deve relatar, por meio do formulário PIP, uma proposta de melhoria de seu processo pessoal e, nesse sentido, o *Scrum Master*, na qualidade de treinador, se compromete a orientar o *desenvolvedor* a atingir as suas metas de melhoria.

O agendamento pessoal do *desenvolvedor* é baseado no tempo estimado para cada tarefa e também na quantidade de horas disponíveis por dia. Com essas informações, o *desenvolvedor* se torna capaz de alocar a quantidade de dias necessários para a realização de todas as tarefas propostas no *Sprint*.

As atividades de acompanhamento do *Sprint* são contempladas pelas etapas 4 a 9 da Estratégia PCU|$_{PSP.}$ Cada uma dessas etapas fornece subsídios para que o *Scrum Master*, que correspondente ao *gerente* na Estratégia PCU|$_{PSP}$, acompanhe quantitativamente as diversas variáveis envolvidas no desenvolvimento. A etapa 5 fornece ao *desenvolvedor* informações sobre seu desempenho e estas podem ser repassadas para o *Scrum Master*. Nas etapas 6 e 8, o *Scrum Master* tem condições de fazer uma avaliação quantitativa dos Fatores Ambientais, tais como: dificuldade com a linguagem e nível de conhecimento em orientação a objetos. Essas informações podem ser utilizadas em um possível replanejamento do projeto. Já nas etapas 7 e 9 o *Scrum Master* recalcula o NDE de cada *desenvolvedor*, e com essa medida é possível fazer um acompanhamento quantitativo do progresso do projeto. Todas as avaliações feitas nas etapa s de 5 a 9 fornecem subsídios para as reuniões diárias prevista no método *Scrum*.

O gráfico *Burndown* (Figura 5.8) é um instrumento de acompanhamento da produção dos Desenvolvedores, muito utilizado no método *Scrum* [*Scrum*, 2007], que mostra quantas horas ou pontos faltam para a realização do *Sprint* a cada dia. Com a

aplicação da Estratégia, o gráfico é montado a partir da Reunião de Planejamento, entre o *gerente* (*Scrum Master*) e os Desenvolvedores (*Team*) em que é definido o total de horas ou Pontos por Caso de Uso necessários para a conclusão do *Sprint*. O replanejamento (etapa 2) é feito a cada mudança de Fatores Ambientais ou na conclusão de um Caso de Uso e o gráfico, então, será atualizado.

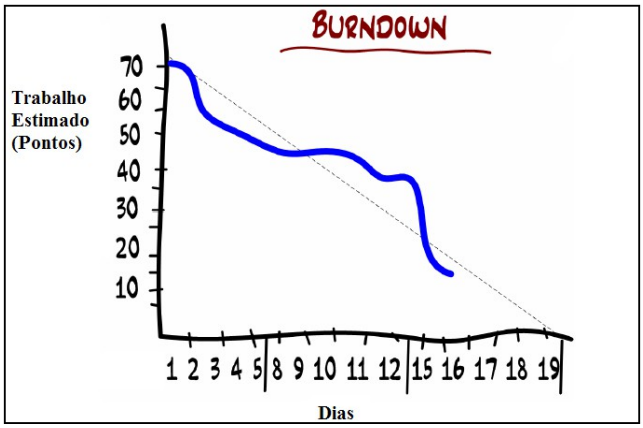


Figura 5.8 - Gráfico Burndown (adaptado de [*PEREIRA*,2007])

Finalmente, na atividade de Retrospectiva é feita uma revisão de todas as atividades e tarefas realizadas ao longo de todo processo e é averiguado o que funcionou e o que precisa ser melhorado. Nesse sentido, a estratégia $PCU|_{PSP}$ fornece uma grande contribuição, na medida em que ela possibilita a análise de desempenho, em termos quantitativos, de cada *desenvolvedor*.

O *Scrum Master* atua como facilitador nas reuniões diárias previstas no método *Scrum*, e torna-se o responsável por remover quaisquer obstáculos que sejam levantados pela equipe nessas reuniões. A lista desses impedimentos é conhecida com *Impediment Backlog*. A Estratégia não trata especificamente de obstáculos, e nem de gerência de riscos. Mas é esperado que nas reuniões de Planejamento e Controle o consenso entre *gerente* e *desenvolvedor* permita a ambos compreender os caminhos que removam os obstáculos.

Conforme citado no início desta seção, o relacionamento entre o método *Scrum* e a Estratégia $PCU|_{PSP}$ é muito natural. A estratégia proposta define detalhes de

implementação que não são tratados pelo *Scrum*, entretanto, a estratégia não estabelece nenhum processo completo de desenvolvimento, ou seja, um processo que abranja desde a fase de elaboração requisitos até a fase de entrega e manutenção do produto final. Assim, de acordo com essa análise apresentada, observa-se que a estratégia proposta e o método *Scrum* exibem características distintas, mas complementares entre si.

5.5 O MPS-BR e a Estratégia PCU|$_{PSP}$

Como foi mencionado no Capítulo 2, o MPS.BR é o primeiro modelo de qualidade de processo de software desenvolvido no Brasil, adequando-se à realidade de pequenas e médias empresas nacionais [SOFTEX,2007].

Conforme guia de implementação do MPS.BR de 2007, as empresas estão divididas em sete níveis de capacitação de qualidade, representados do nível G ao nível A. Atualmente há no Brasil 48 empresas com certificação MPS.BR, sendo que 30 se encontram no nível G, 08 no nível F, 5 no nível E, 1 no nível D, 0 no nível C, 0 no nível B e 4 no nível A. Há também 78 empresas com certificação CMMI, 51 no nível 02, 17 no nível 3 e 10 no nível 05 [SANTOS,2007].

A Estratégia PCU|$_{PSP}$ contribui para a implantação dos processos do MPS.BR, mostrados na Figura 5.9.

O nível B foca o desenvolvimento em Gerência de Projetos, por meio de um modelo quantitativo/estatístico. A Estratégia contribui para a implantação do Nível B, através da gerência quantitativa do NDE. A estratégia PCU|$_{PSP}$ provê uma maneira de, constantemente, fazer uma gerência quantitativa do projeto por meio do controle e atualização do valor do NDE, permitindo manter uma estimativa sempre atualizada para a duração do projeto como um todo.

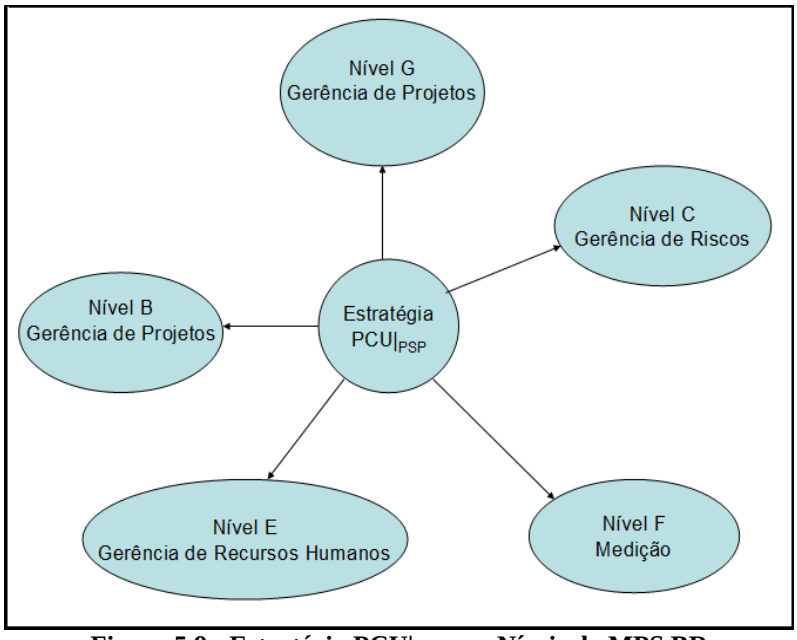


Figura 5.9 - Estratégia PCU|$_{PSP}$ e os Níveis do MPS.BR

No que diz respeito ao Nível C, a estratégia PCU|$_{PSP}$ pode dar suporte a gerência de alguns tipos de riscos, principalmente no que se refere aos problemas encontrados pela equipe de desenvolvimento, que serão apontados rapidamente pelo uso do PSP, e aos Fatores de Complexidade Técnica e Ambientais que estarão constantemente sendo monitorados pela aplicação do PCU juntamente com o PSP.

A Estratégia aborda dois processos do nível E: Gerência de Recursos Humanos e Gerência de Projetos. No primeiro, a Estratégia provê uma forma de avaliação dos Desenvolvedores de forma objetiva, dando condições ao *gerente* de avaliar o nível de conhecimento, através dos Fatores Ambientais, e propor treinamento aos integrantes da equipe. Nesse sentido, a proposta recomenda ao *gerente* uma postura diferenciada: ele se relaciona com os Desenvolvedores como um Treinador, buscando uma melhoria na produtividade dos Desenvolvedores. A Gerência de Projetos do Nível E propõe uma intersecção com outros processos, dentre os quais o processo de medição, e a Estratégia facilitam a medição de diversos produtos de trabalho.

No nível F, o processo de Medição é apoiado pela Estratégia, uma vez

que esta se baseia na aplicação constante do PSP e, em decorrência disso, a disciplina de registrar tempo e tamanho das rotinas, prevista no PSP, bem como a medição e avaliação constante do plano do projeto como um todo, dão suporte à atividade de medição, tanto do ponto de vista de artefatos como do processo em si.

5.6 Análise conjunta do MPS-BR, Scrum e a Estratégia PCU|PSP

Como o foco dessa dissertação se refere ao processo de gerência de projetos, avalia-se nesta seção, a Estratégia PCU|PSP, juntamente com o Método *Scrum*, de acordo com os atributos de qualidade do processo de Gerência de Projetos, previstos no modelo MPS.BR, nível G. Enquanto a Metodologia *Scrum* analisa o processo de forma mais abrangente, partindo dos requisitos à entrega do produto final, a Estratégia PCU|PSP enfoca o planejamento e o controle do trabalho dos Desenvolvedores no desenvolvimento do software.

No plano de desenvolvimento do MPS.BR, o nível G enfoca o mínimo necessário de capacitação de uma empresa nos processos de Gerência de Projetos e de Gerência de Requisitos. Analisou-se apenas este nível porque é aquele em que se inicia o processo de organização. Nessa fase, todos os processos da empresa são redefinidos, requisitando um maior esforço, pois a empresa passa do nível caótico para o parcialmente gerenciado.

O modelo de qualidade MPS.BR possui mecanismos para avaliar a capacidade do processo, que é representada por um conjunto de Atributos de Processo (AP), descrito em termos de resultados esperados, que expressa o grau de refinamento e institucionalização com que o processo é executado na organização [MPS.BR,2007]. No nível G, o processo Gerência de Projetos deve atender aos atributos de processo AP 1.1 e AP 2.1, descritos no Capítulo 2.

Os atributos de processo AP 2.1 não foram detalhados neste trabalho,

pois ainda não se têm resultados para avaliar consistentemente os efeitos da aplicação da Estratégia PCU|PSP em relação a esses atributos de processo em uma pequena empresa.

A Tabela 5.3, exibe todos os itens referidos como resultados esperados do processo Gerência de Projetos relacionados ao atributo “AP 1.1 - Processo é executado”, que define o quanto o processo atinge seu propósito, e estabelece uma relação entre os resultados esperados no MPS.BR e os resultados obtidos com o emprego da Estratégia PCU|PSP, aplicada em conjunto com a metodologia *Scrum*.

Tabela 5.3 - Estratégia PCU|PSP , *Scrum* e o MPS.BR

Resultado esperado no MPS.BR	**Indícios dos resultados alcançados pela Estratégia PCU|PSP juntamente com o *Scrum***
GPR 1 - O escopo do trabalho para o projeto é definido.	Com a aplicação do *Scrum*, define-se o escopo do trabalho na atividade Planejamento 1 do *Sprint*, em uma reunião, onde o *Product Owner* discutirá com a equipe a lista de itens do *Product Backlog*, responsabilizando-se, a partir desse momento por ela. Salienta-se que o processo *Scrum*, por ser um método ágil, parte do pressuposto que nem todos os aspectos do projeto são previsíveis em seu início. A Estratégia PCU|PSP prevê uma reunião de planejamento, que no *Scrum* corresponde ao Planejamento 2 do *Sprint*, com o objetivo de planejar o trabalho de desenvolvimento e reavaliar o entendimento dos itens do *Product Backlog* (Casos de Uso). Nessa reunião, podem ser encontrados itens que pertencem ao escopo, mas que não foram previstos.
GPR 2 - As tarefas e os produtos de trabalho do projeto são dimensionados utilizando métodos apropriados.	Uma vez definido o escopo, o *Scrum Master*, baseando-se na Estratégia PCU|PSP, realiza uma estimativa de esforço para cada Caso de Uso, e através do NDE, calcula o número de horas necessárias para o desenvolvimento de cada Caso de Uso. Após isso, os Desenvolvedores decompõem os Casos de Uso em cenários e rotinas, e esses produtos de trabalho são dimensionados utilizando o processo PSP, propiciando aos Desenvolvedores um planejamento do seu trabalho, por meio da organização de seu cronograma individual.
GPR 3 - O modelo e as fases do ciclo de vida do projeto são definidas.	A Estratégia associada ao processo *Scrum* ajuda a definir muitas fases do ciclo de vida do projeto, bem como os produtos de trabalho, que serão utilizados de uma fase para outra e também os marcos necessários para o controle do projeto. Assim, observam-se as fases: 1. *Product Backlog*, Planejamento 1 do *Sprint*, Planejamento 2 do *Sprint*: Planejamento de Alto Nível, Planejamento Detalhado de Rotinas, *Sprint Backlog*, *Sprint*: Codificação, Testes, Correção, Ajuste de Planejamento Pessoal (Rotina Completa, Cenários Completos e Casos de Uso Completos), Ajuste de Fatores Ambientais (Fatores Ambientais) e Ajuste de NDE (Cálculo de NDE Atual), Retrospectiva*:* Replanejamento de Alto Nível.
GPR4 - (Até o nível F) O esforço e o custo	A Estratégia PCU|PSP propõe duas formas de cálculo de estimativa: uma baseada no PCU e a outra baseada no PSP. O cálculo do NDE,

para a execução das tarefas e dos produtos de trabalho são estimados com base em dados históricos ou referências técnicas.	baseado no PCU, fundamenta-se no cálculo do NDE médio, portanto, para realizá-lo, utilizam-se referências técnicas. Observa-se, por outro lado, que o cálculo baseado no PSP, para o planejamento pessoal do *desenvolvedor*, é estimado com base em dados históricos, visto que ele prevê o tempo de uma tarefa considerando o registro de tempo das tarefas anteriores. Com o retorno do *desenvolvedor* aos planejamentos anteriores, ele pode verificar o motivo que o levou a errar as estimativas passadas, imprimindo a ele uma contínua melhoria na produção de seu planejamento.
GPR5 - O orçamento e o cronograma do projeto, incluindo marcos e/ou pontos de controle, são estabelecidos e mantidos.	A Estratégia PCU|$_{PSP}$ fornece um conjunto de estimativas que são utilizadas para a montagem do cronograma de cada *desenvolvedor*. Os pontos de controle são estabelecidos mediante a conclusão de produtos de trabalho, tais como: rotinas, cenários, casos de uso e *Sprints*.
GPR6 - Os riscos do projeto são identificados e o seu impacto, probabilidade de ocorrência e prioridade de tratamento são determinados e documentados.	Na metodologia *Scrum*, o *Scrum Master* é o responsável por identificar, registrar e avaliar os riscos (*Impediment Backlog*), assim como buscar alternativas de solução para cada um deles. Nas reuniões, realizadas pelo *Scrum Master*, com o *Team* e com o *Product Owner*, é organizada uma lista contendo os impedimentos e obstáculos a serem suplantados, e é de responsabilidade do *Scrum Master* solucionar todos os problemas encontrados, a fim de que o produto final seja entregue com qualidade e dentro dos prazos estabelecidos. Na reunião de planejamento o *gerente* pode provocar a discussão de possíveis riscos para poder entrar em um consenso com os Desenvolvedores.
GPR7 - Os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo.	Os papéis são definidos em *gerente* (*Scrum Master)*, *Product Owner* e Desenvolvedores (*Team*), de acordo com a necessidade de cada projeto. O *gerente* tem capacidade de liderança para administrar sua equipe, segundo o estilo determinado pela Estratégia PCU|$_{PSP}$. Já o *Product Owner* é o membro da equipe que possui conhecimento mais especializado acerca do produto a ser desenvolvido, por isso ele é o responsável por definir os requisitos do produto. Os Desenvolvedores são aqueles que constroem o produto. São avaliados constantemente pelo *gerente*, considerando-se os Fatores Ambientais, descritos na Tabela 3.2 definidos na Estratégia PCU|$_{PSP}$.
GPR8 - As tarefas, os recursos e o ambiente de trabalho necessários para executar o projeto são planejados.	Na Estratégia PCU|$_{PSP}$ associada à Metodologia *Scrum*, após a seleção de Casos de Uso, na reunião Planejamento 2 do *Sprint* (Figura 5.5), realiza-se o planejamento das tarefas. Quando a tarefa corresponde ao desenvolvimento de uma rotina do programa, o planejamento das tarefas é feito pela decomposição dos Casos de Uso em cenários, e posteriormente os cenários em rotinas. Com relação aos recursos, a Estratégia se limita a fazer a alocação da equipe de desenvolvimento, bem como dimensionar corretamente a quantidade de Desenvolvedores para o cumprimento dos prazos. Finalmente sobre o ambiente de trabalho, a Estratégia estabelece um procedimento de avaliação continuada dos Desenvolvedores, e através do papel de treinador, o *gerente* aprimora a performance da sua equipe.
GPR9 - Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição.	Os dados relevantes do projeto que são planejados quanto à forma de coleta, armazenamento e distribuição correspondem aos itens do *Product Backlog, Sprint Backlog, Impediment Backlog,* e dados de produtividade e produção de cada *desenvolvedor*. Não é o foco deste trabalho estabelecer formas de acesso aos dados, mas espera-se que nas reuniões de planejamento as questões pertinentes aos dados

Um mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança.	sejam solucionadas.
GPR10 - (Até o nível F) Planos para a execução do projeto são estabelecidos e reunidos no Plano do Projeto.	Não é o foco deste trabalho definir como os Planos para execução do projeto serão estabelecidos e reunidos no Plano do Projeto. Porém, nas reuniões de Planejamento 1 e 2 do *Sprint* são definidos os cronogramas de trabalho e o planejamento de alto nível, que compõem o Plano de Projeto.
GPR11 - A viabilidade de atingir as metas do projeto, considerando as restrições e os recursos disponíveis, é avaliada. Se necessário, ajustes são realizados.	A viabilidade das metas é avaliada constantemente. De acordo com a Estratégia, os Desenvolvedores analisam o seu desempenho ao término de cada rotina. Com a realização de um cenário completo, os Fatores Ambientais são avaliados e, se necessário, o planejamento é reajustado. Quando um Caso de Uso é finalizado, é feito o cálculo do NDE e, se necessário, o planejamento será refeito. Por fim, quando um *Sprint* é finalizado, é organizada uma reunião entre o *Scrum Master* e o *Team*, onde é realizada uma retrospectiva do trabalho e as metas são reavaliadas.
GPR12 - O Plano do Projeto é revisado com todos os interessados e o compromisso com ele é obtido.	Nas reuniões conhecidas como Planejamento 1 e 2 do *Sprint* são revisados os Planos do Projeto como também é obtido o compromisso dos interessados. Nestas reuniões tarefas são alocadas aos Desenvolvedores, que assumem suas atribuições perante a equipe (*Team*). Outros membros do grupo de interessados são comprometidos com o Plano de Projeto, ao final de cada *Sprint*, inclusive o cliente final, pois, estes deverão testar e validar as novas funcionalidades.
GPR13 - (Até o nível F) O progresso do projeto é monitorado com relação ao estabelecido no Plano do Projeto e os resultados são Documentados.	A Estratégia PCU|$_{PSP}$ estabelece um sistema de monitoramento refletido nas reuniões diárias exigidas pelo *Scrum*, nas quais o *Scrum Master* acompanha todo o progresso do projeto, bem como a produtividade de cada *desenvolvedor*. O método *Scrum* pressupõe equipes auto-gerenciadas portanto, tanto o *Scrum Master* quanto o *Team* são responsáveis por avaliar o progresso da equipe em relação ao Plano do Projeto e tomar as devidas providências caso haja distorção.
GPR14 - O envolvimento das partes interessadas no projeto é gerenciado.	O envolvimento de todas as partes interessadas no projeto é gerenciado nas reuniões previstas no método *Scrum*, como também nas reuniões de Planejamento e Controle da Estratégia PCU|$_{PSP}$. Salienta-se que nas reuniões diárias, o *Scrum Master* tem condições de avaliar o envolvimento das partes interessadas através do progresso do projeto.
GPR15 - Revisões são realizadas em marcos do projeto e conforme estabelecido no planejamento.	Os marcos do projeto de maior relevância são: quando termina uma rotina, um cenário, um Caso de Uso ou um *Sprint*. Quando termina uma rotina, o *desenvolvedor* faz revisões no seu planejamento pessoal. Quando termina um cenário ou um Caso de Uso, o *gerente* faz revisões no planejamento do projeto. Quanto termina um *Sprint* o método *Scrum*, determina que seja realizada uma reunião em que é feita uma retrospectiva do trabalho, onde se realiza revisões no planejamento do projeto.
GPR16 - Registros de problemas identifica-	O método *Scrum* prevê o registro de problemas conhecido como "*Impedimet Backlog*", que serão analisados diariamente pelo *Scrum*

dos e o resultado da análise de questões pertinentes, incluindo dependências críticas, são estabelecidos e tratados com as partes interessadas.	*Master,* que é o responsável por solucionar os problemas encontrados. Nas reuniões de Planejamento e são definidos os meios pelos quais os problemas serão resolvidos.
GPR17 - Ações para corrigir desvios em relação ao planejado e para prevenir a repetição dos problemas identificados são estabelecidas, implementadas e acompanhadas até a sua conclusão.	A Estratégia propõe reuniões periódicas de controle do andamento do projeto, e ações para corrigir os desvios em relação ao planejado são realizadas. Caso haja um grande desvio em relação ao planejado, está previsto na Estratégia o replanejamento do projeto. Para prevenir a repetição de problemas identificados a Estratégia propõe o uso do PIP, uma atividade do PSP, em que o *desenvolvedor* propõe uma melhoria em suas atividades, quando um problema aparece. Também está previsto a lista chamada *Impediment Backlog* que *Scrum Master* deve buscar solucionar todos os itens. Uma importante atividade proposta pelo método *Scrum* é a retrospectiva, onde se realiza revisões no planejamento do projeto.

5.4 Considerações Finais

Apresentou-se neste capítulo a Estratégia PCU|$_{PSP}$, proposta neste trabalho, cujo objetivo é apoiar o planejamento do desenvolvimento de software por meio do uso conjunto das técnicas Pontos de Casos de Uso (PCU) e *Personal Software Process* (PSP). A definição dessa estratégia surgiu do uso dessas técnicas na prática em duas pequenas empresas que buscam a melhoria de seus processos e maior agilidade no desenvolvimento dos produtos.

A sistematização do uso dessas técnicas na definição de uma estratégia levou a uma abordagem que apóia o *gerente* na elaboração de um plano de desenvolvimento, na distribuição das tarefas, na estimativa de tempo e esforço e, principalmente, no acompanhamento do desenvolvimento, podendo identificar problemas e tomar medidas corretivas de forma bastante rápida.

O uso dos Pontos por Casos de Uso permite uma distribuição bastante apropriada das funcionalidades a serem implementadas e o uso do PSP permite uma auto avaliação e um auto planejamento ao *desenvolvedor*, bem como uma realimentação bastante precisa e em tempo para o *gerente*.

Uma vez definida a estratégia pôde-se observar que ela se adequava muito naturalmente ao processo do método ágil *Scrum*, que foi o processo escolhido pelas empresas para dar maior agilidade ao desenvolvimento. Dessa forma, apresentou-se também neste capítulo uma análise de como a estratégia $PCU|_{PSP}$ se adequa a esse processo.

Além disso, apresentou-se também uma análise de como o uso da estratégia pode facilitar a implantação do modelo de qualidade de processo MPS.BR, no que diz respeito às áreas de processo diretamente relacionadas ao foco principal da estratégia, que é o planejamento e o acompanhamento do projeto de desenvolvimento de software. Como foi mostrado, o uso da estratégia impõe uma sistemática de trabalho que acaba por satisfazer vários requisitos exigidos por esse modelo.

No próximo capítulo serão apresentados estudos de casos que mostram o uso da estratégia na prática, nas pequenas empresas aqui consideradas.

Capítulo 6 - Estudo de Caso

6.1 Considerações iniciais

Nesta seção são relatados três estudos de caso de aplicação da estratégia PCU|$_{PSP}$. Nos dois primeiros, a ênfase é dada no controle do projeto de software, através dos constantes ajustes do NDE, evidenciando, portanto, as vantagens de se combinar a técnica PSP com o método PCU, pois essa agregação das duas técnicas proporciona uma forma muito apropriada de se fazer o planejamento e o controle desse planejamento durante o desenvolvimento do software. No terceiro estudo de caso, a ênfase é dada no gerenciamento dos desenvolvedores, por meio da análise da sua produtividade e suas limitações, pelo fato deles utilizarem o PSP.

O capítulo está organizado da seguinte forma: na seção 6.2 é apresentada a caracterização das empresas utilizadas nos estudos de caso. Na seção 6.3 é apresentado o estudo de caso na empresa Linkway, no qual foi desenvolvido um portal na *Web*. Na seção 6.4, é apresentado o estudo de caso na empresa NBS, no qual foi desenvolvido um sistema *desktop* para automação das atividades de uma secretaria de Câmara Municipal. Na seção 6.5, apresenta-se um estudo que mostra os resultados da aplicação do processo PSP com a finalidade de auxiliar o planejamento e o acompanhamento de um projeto de software e finalmente, na seção 6.6 apresenta-se as considerações finais.

6.2 Caracterização das Pequenas Empresas Utilizadas no Estudo de Caso.

As empresas nas quais as estratégias foram definidas e vem sendo utilizada, apresentadas aqui como as empresas dos estudos de caso, foram a Linkway e a

NBS. A Linkway é um pequeno provedor de Internet que desenvolve aplicativos *Web*, e a NBS, uma pequena empresa de software, focada em aplicativos de gestão e administração pública. Ambas são situadas na cidade de São Carlos, interior do Estado de São Paulo.

A Linkway tem uma equipe de desenvolvimento formada por um *gerente*, dois consultores de vendas, dois *Web Designers* e três Desenvolvedores Java. A empresa está há dez anos no mercado desenvolvendo aplicativos *Web* sob encomenda. A estratégia de mercado de não desenvolver uma linha de produtos de uso genérico, mas sim, de construir aplicativos baseados nas necessidades de cada cliente, criou um ambiente de desenvolvimento de intensa produção de software, com as mais variadas necessidades. O controle de prazos e custo é, portanto, vital, uma vez que a receita está diretamente vinculada à entrega de cada sistema dentro do custo e do prazo estipulados. Assim, a cada novo sistema, um novo ciclo de desenvolvimento é criado. A necessidade de resultados rápidos – uma característica de sistemas *Web* [CAPILLA, 2003] – criou um ambiente favorável de melhoria contínua nos processos de desenvolvimento dos projetos, que foram fundamentais para a formulação da estratégia proposta nesta dissertação.

A NBS é uma empresa que participa de um mercado com relativa estabilidade de requisitos, pois as regras de negócio de administração pública não mudam com a mesma intensidade de outros setores. O desenvolvimento está concentrado em construir novos módulos aos aplicativos já existentes, com a finalidade de acrescentar novas funcionalidades e, assim, tornar o sistema mais abrangente. O desenvolvimento se concentra em um ambiente *Desktop* ao invés de tecnologias *Web*. A equipe de desenvolvimento da NBS é formada por um *gerente*, dois Desenvolvedores, dois analistas de suporte e um consultor de vendas.

As duas empresas utilizam o paradigma de Orientação a Objetos e, para a modelagem dos sistemas, a UML – *Unified Modeling Language* [LARMAN,2004]. Ainda, ambas as empresas utilizam protótipos de tela parcialmente funcionais, a fim de definir bem os requisitos.

Há vários anos as duas empresas estão empenhadas em melhorar seus processos de desenvolvimento com vistas aos modelos de qualidade propostos na literatura, tendo inicialmente se dedicado ao estudo e melhoria de seus processos com base no CMMI [CMMI,2006] e, mais recentemente, com a proposta do MPS.BR [MPSBR,2007], pelo fato desse modelo ser mais adequado às empresas de pequeno porte, tem-se procurado atender às suas exigências.

6.3 Estudo de Caso 1: Empresa Linkway – Sistema Tapetes

Este estudo de caso foi baseado no desenvolvimento de um portal *Web* para uma indústria de tapetes, com as seguintes funções: um catálogo dos produtos fabricados, um demonstrativo dos representantes, notícias pertinentes aos produtos e outras informações institucionais sobre a empresa. As informações contidas nesse portal são armazenadas em um banco de dados e são totalmente gerenciadas pela *Web*, através do próprio sistema, aqui denominado Sistema Tapetes.

O desenvolvimento do código em linguagem Java consumiu aproximadamente 216 horas, distribuídas pelos nove Casos de Uso apresentados na Tabela 6.1. Junto aos Casos de Uso, apresentam-se as respectivas complexidades, que totalizam 95 pontos para o sistema todo. Esse sistema possui dois atores complexos totalizando 101 pontos não ajustados para o sistema. As complexidades dos Casos de Uso foram definidas conforme a proposta de Probasco [PROBASCO,2002], baseada na identificação de cenários e não em transações atômicas como sugere Karner [KARNER,1993]. Essa aplicação foi inteiramente desenvolvida por apenas um

desenvolvedor.

Tabela 6.1 - Casos de Uso do Sistema Tapetes

	COMPLEXIDADE			
CASOS DE USO	**Simples (5)**	**Médio (10)**	**Complexo (15)**	**Pontos**
Representantes			X	15
Contato	X			5
Notícias		X		10
Catalogo			X	15
Ambientes		X		10
Administração Geral	X			5
Currículo		X		10
Revenda		X		10
Internauta			X	15
TOTAL				**95**

É importante, neste momento, fazer algumas considerações sobre a implantação da estratégia proposta nesta dissertação, para a realidade de uma pequena empresa. Um projeto de aproximadamente 200 horas, é considerado de porte pequeno, pois fazendo um cálculo básico, em que um desenvolvedor é capaz de trabalhar, em média, 5 horas por dia, esse projeto teria a duração de quase dois meses. Por outro lado, Karner propõe um NDE de 20 horas homem para cada Ponto por Caso de Uso. Probasco afirma que realmente esse valor pode variar entre 20 e 30 em grandes empresas, como a IBM e a SUN MicroSystem [PROBASCO,2002]. Entretanto, com o NDE nesse patamar, um Caso de Uso médio (10 pontos) pode consumir até 300 horas (10 x 30), o que faria com que a previsão de tempo de desenvolvimento do sistema em questão, para os nove Casos de Uso, alcançasse o valor de 3030 horas (101 x 30), um número muito elevado quando comparado ao valor de 216 horas, que foi realmente consumido.

Percebe-se, na prática das pequenas empresas analisadas nesta dissertação, que o NDE deve ser ajustado para a realidade de cada empresa. A diferença discrepante da quantidade de horas observadas se deve, principalmente, à granularidade assumida para cada Caso de Uso. Uma pequena empresa, que trabalha com projetos

menores, deve assumir uma granularidade menor de Caso de Uso para ter um detalhamento maior do sistema. Por outro lado, uma grande empresa que trabalha em grandes projetos deve assumir uma granularidade maior de cada Caso de Uso, pois quanto maior o sistema, menor deve ser o detalhamento da análise.

Outro ponto a destacar que influencia a diferença entre o nível de esforço proposto por Karner [KARNER,1993] e ajustado neste estudo de caso é o fato de que o *gerente* da estratégia $PCU|_{PSP}$ não considera todas as atividades do processo de desenvolvimento para efeito do cálculo da previsão do esforço em horas. Nesta proposta, o cálculo é feito no momento em que os Casos de Uso já foram elaborados, e também, não computa o tempo de implantação. A proposta de Karner [KARNER,1993] leva em consideração todas as atividades envolvidas no processo de desenvolvimento.

Na Linkway, através da interação entre *gerente* e *desenvolvedor*, foram definidos o Fator de Complexidade Técnica, o Fator Ambiental, o primeiro valor do NDE e a estimativa inicial de horas para o desenvolvimento, de acordo com a técnica de Pontos por Casos de Uso.

O Fator de Complexidade Técnica foi calculado em 1,10, conforme o postulado por Karner [KARNER,1993] no Capítulo 3 desta dissertação, pois apenas alguns dos seus fatores tiveram influência no sistema desenvolvido, como descrito na Tabela 6.2, e os demais fatores não tiveram influência no sistema e, portanto, foram fixados em 3, que é um valor médio (se todos os fatores fossem iguais a 3 o FCT seria aproximadamente igual a 1). Na Tabela 6.2, temos: Descrição, Peso, Influência, Fator e o Motivo pelo qual o valor da influência foi fixado.

O Fator Ambiental foi calculado em 0,85, analogamente ao Fator de Complexidade Técnica, em que alguns fatores tiveram influência e outros foram fixados no valor médio de 3, como proposto por Karner [KARNER,1993]. Na Tabela 6.3, estão

relacionados os fatores que contribuíram para o cálculo do Fator Ambiental.

Tabela 6.2 - Cálculo do Fator de Complexidade Técnica do Sistema Tapetes

FCT	Descrição	Peso	Influência	Fator	Motivo
F1	Sistema distribuído	2	3	6	Valor médio (3)
F2	Tempo de Resposta	1	3	3	Valor médio (3)
F3	Eficiência	1	3	3	Valor médio (3)
F4	Processamento complexo	1	3	3	Valor médio (3)
F5	Código reusável	1	5	5	Todo desenvolvimento da Linkway é feito para ser reutilizado.
F6	Facilidade de instalação	0,5	3	1,5	Valor médio (3)
F7	Facilidade de uso	0,5	4	2	Em sistemas para Web é dada grande atenção à facilidade de uso.
F8	Portabilidade	2	4	8	Os sistemas devem ser desenvolvidos independentes de plataforma e navegador.
F9	Facilidade de mudança	1	4	4	A manutenibilidade é um fator de qualidade de software da Linkway.
F10	Concorrência	1	3	3	Valor médio (3)
F11	Recursos de segurança	1	5	5	Em sistemas para Web segurança é um requisito primordial.
F12	Acessível por terceiros	1	3	3	Valor médio (3)
F13	Requer treinamento especial	1	3	3	Valor médio (3)
FCT = 1,10					

Tabela 6.3 - Cálculo do Fator Ambiental

FCT	Descrição	Peso	Influência	Fator	Motivo
E1	Familiaridade com o processo de desenvolvimento.	1,5	4	6	*Desenvolvedor* com alguns anos de experiência com programação na Linkway.
E2	Desenvolvedores em meio expediente.	-1	3	-3	Valor médio (3)
E3	Capacidade em Análise	0,5	4	2	*Desenvolvedor* com experiência em análise de sistemas.
E4	Experiência com a aplicação em desenvolvimento.	0,5	5	2,5	Desenvolveu aplicações semelhantes.
E5	Experiência em Orientação a Objetos.	1	4	4	*Desenvolvedor* com experiência em Orientação a Objetos.
E6	Motivação	1	4	4	Informado pelo *desenvolvedor*
E7	Dificuldade com a Linguagem de programação	-1	3	-3	Valor médio (3)
E8	Requisitos Estáveis	2	3	6	Valor médio (3)
FA = 0,85					

Na tabela 6.4, descrevem-se os valores iniciais de planejamento para Fatores de Complexidade Técnica, Fatores Ambientais, Pontos por Casos de Uso Ajustados, NDE inicial, Tempo Total Estimado, e os valores finais relativos ao Tempo Total Efetivamente Consumido e o Erro em porcentagem na Estimativa, observados

depois do término do sistema.

Tabela 6.4 - Valores Iniciais de Planejamento e Valores Finais do Sistema Tapetes

Descrição	Valor
Fator de Complexidade Técnica	1,10
Fator Ambiental	0,85
Pontos por Caso de Uso Ajustados	93,45
NDE Inicial (Dado Histórico da Empresa)	3 horas homem
Tempo Total Inicialmente Estimado	280,40 horas
Tempo Total Efetivamente Consumido	216 horas
Erro na Estimativa	29,69%

Os Pontos por Caso de Uso ajustados foram calculados e totalizaram 93,45. Baseado neste resultado, calculou-se o tempo estimado para o desenvolvimento do aplicativo, multiplicando os Pontos por Caso de Uso (93,45) pelo NDE inicial, com valor histórico 3, já apresentado por este *desenvolvedor* em projetos anteriores. Dessa forma, a previsão para o desenvolvimento do sistema foi de 280,40 horas. Como o sistema foi desenvolvido em 216 horas, esta estimativa apresentou um erro, em relação ao valor estimado, de 29,69 % ou de 64,40 horas o que não inviabiliza essas previsões, diante de um cálculo de estimativa passível de negociações.

Na Tabela 6.5, apresentam-se os nove Casos de Uso do sistema e o resultado da aplicação da estratégia $PCU|_{PSP}$. Cada linha dessa tabela apresenta, em relação a cada Caso de Uso que compõe o sistema, a complexidade do Caso de Uso, os Pontos por Caso de Uso ajustados (individual e acumulado), as horas efetivamente consumidas para conclusão do Caso de Uso (individual e acumulada), o NDE, que corresponde ao nível de esforço real para seu desenvolvimento, o estimado em horas para o desenvolvimento do sistema (re-estimado a cada novo valor do NDE), e a diferença em horas em relação ao planejamento inicial de 280,40 horas.

Assim, para cada Caso de Uso que é concluído, calculam-se os valores acumulados dos Pontos por Casos de Uso que já foram desenvolvidos e de horas que foram gastas para desenvolvê-los. Tomando-se como exemplo o Caso de Uso 2, depois

de calculados os valores acumulados, calcula-se o novo NDE, dividindo-se as horas acumuladas (32,96) pelos Pontos por Casos de Uso acumulados (19,74) obtendo-se o valor 1,67. Esse valor significa que o nível de esforço para realização desse Caso de Uso não foi igual ao obtido para realização do Caso de Uso 1, calculado em 1,70. Com esse novo nível de esforço, calcula-se novamente a quantidade de horas para desenvolvimento do sistema todo (156,07), multiplicando-se o novo NDE pelo total de Pontos por Casos de Uso do sistema (93,45). Isso significa que se o desenvolvimento transcorrer dessa forma desse momento em diante, o planejamento inicial possui um erro em relação ao que efetivamente está acontecendo de 124,29 horas. Observando-se o Caso de Uso 9, percebe-se que o NDE subiu para 2,31 e que o total efetivo de horas gastas para desenvolver o sistema todo foi de 216,21 horas.

Tabela 6.5 - Resultado da Aplicação da Estratégia PCU|$_{PSP}$ no Sistema Tapetes

	Etapa 1	Etapas 1 e 2		Etapa 4		Etapa 9	Etapa 2	Etapa 2
Caso de Uso	Complexidade de cada Caso de Uso	Pontos por Caso de Uso ajustados		Horas Efetivamente Consumidas para conclusão do Caso de Uso		NDE	Estimado (horas)	Diferença em horas em relação ao planejamento inicial de 280,40 horas
		Individual	Acumulado	Individual	Acumulado			
1	Complexo	14,50	14,50	24,60	24,60	1,70	158,87	121,49
2	Simples	5,24	19,74	8,36	32,96	1,67	156,07	124,29
3	Médio	9,87	29,61	38,20	71,16	2,40	224,29	56,07
4	Complexo	14,50	44,10	53,50	124,66	2,83	264,47	15,89
5	Médio	9,87	53,97	10,20	134,86	2,50	233,63	46,73
6	Simples	5,24	59,22	6,50	141,36	2,39	223,35	57,01
7	Médio	9,87	69,09	29,50	170,86	2,47	230,83	49,53
8	Médio	9,87	78,96	28,50	199,36	2,52	235,50	44,86
9	Complexo	14,50	93,45	16,85	216,21	2,31	215,88	64,48

Outro ponto a ser observado está relacionado à complexidade dos Casos de Uso. Os Casos de Uso de complexidade simples (2 e 6) consumiram uma quantidade de horas compatível com a esperada, menor que os Casos de Uso de complexidade média. Já os Casos de Uso complexos não apresentaram essa consistência, pois apenas o Caso de Uso 4 consumiu mais horas que os médios. Assim, o fato do Caso de Uso 1,

embora sendo classificado como complexo, consumir menos horas do que os de complexidade média, gerou um erro grande na primeira estimativa. Esse é um ponto que será investigado em trabalhos futuros, e que está relacionado à influência dos Fatores Ambientais na classificação dos Casos de Uso, de acordo com a proposta de Karner. Além de outras variáveis que interferem nessa distorção, o que se pode observar é que à medida que o desenvolvedor se familiariza com o domínio da aplicação e com as técnicas utilizadas, ganhando experiência de uma maneira geral, seu nível de esforço decresce.

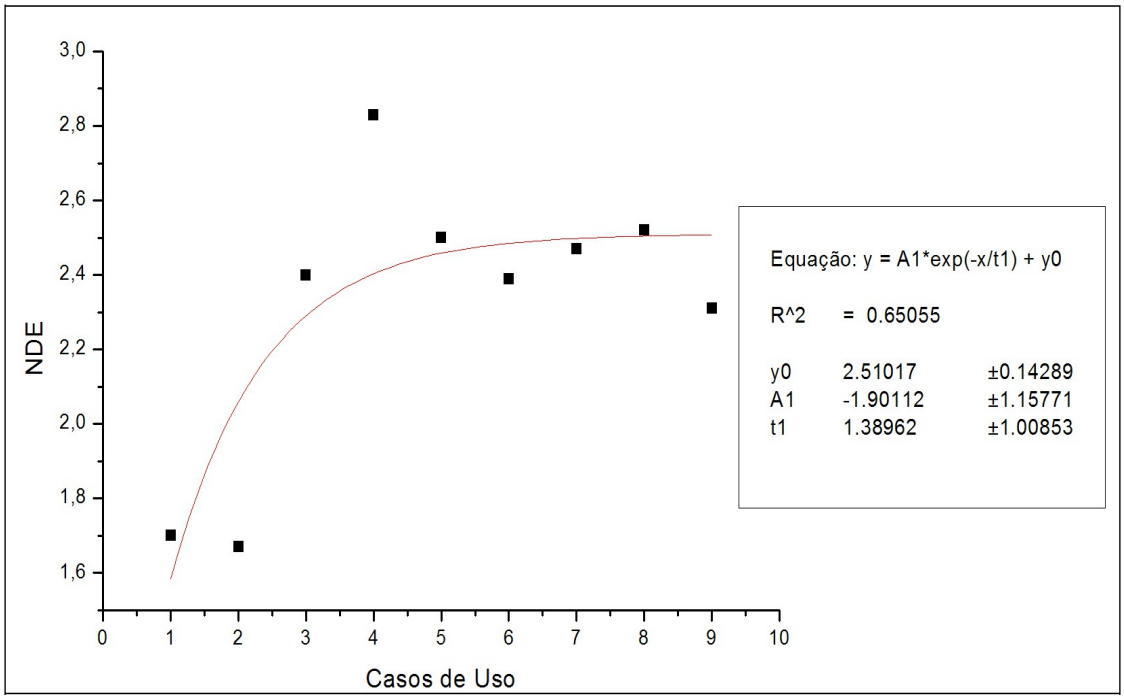
Outro aspecto observado, neste estudo de caso, é que o valor do NDE tende a se estabilizar dentro de um mesmo projeto. Foi ajustada uma função aos valores do NDE da Tabela 6.5, e na Tabela 6.6, tem-se a equação gerada pelo software OriginPro 7.5 [ORIGINPRO,2007] .

Tabela 6.6 - Função ajustada ao NDE

Variável	Valor
y0	2,51017 ± 0,14289
A1	-1,90112 ± 1,15771
t1	1.38962 ± 1,00853
R^2	0,65055
Equação: y = A1 * exp(-x/t1) + y0	

A Figura 6.1 representa um gráfico da equação y = A1 * exp(-x/t1) + y0, onde no eixo x têm-se os Casos de Uso e no eixo Y o NDE , com x sendo o número do Casos de Uso da Tabela 6.5, A1, t1 e y0 constantes, y(x) como o valor do NDE (O gráfico foi gerado no OriginPro 7.5 [ORIGINPRO,2007]).

Figura 6.1 - Gráfico do NDE por Caso de Uso



Para um grande número de Casos de Uso, o termo exponencial tende a zero rapidamente, e o valor extrapolado do NDE não dependerá mais do tempo, como sendo igual a y0. Ou seja, para uma grande quantidade de Casos de Uso, o valor para o qual a grandeza NDE tende é y0. Esta função fornece indícios de que existe uma tendência de estabilização do valor do NDE (2,51017 ± 0,14289). Essa constância possibilitará que o gerente faça previsões mais precisas acerca do planejamento do desenvolvedor.

R^2 é a qualidade do ajuste; quanto mais perto do número 1 melhor é a precisão do ajuste. No entanto, para o estudo de caso em questão, a precisão de R^2 = 0,65055 é suficiente, pois apresenta um erro de apenas ± 0,14289. Observa-se que não há necessidade de ter uma precisão de minutos no desenvolvimento de um Caso de Uso.

6.4 Estudo de Caso 2: Empresa NBS – Sistema Secretaria de Câmaras Municipais

No Estudo de Caso da NBS, foi analisado o desenvolvimento de um módulo novo para um sistema já existente de informatização da Secretaria de Câmaras Municipais. Na Tabela 6.7, apresentam-se os seis Casos de Uso desenvolvidos na construção do módulo. As complexidades desses Casos de Uso totalizam 75 pontos não ajustados. Esse módulo vai ser acessado por dois atores complexos, segundo a metodologia de Karner [KARNER,1993], perfazendo mais seis pontos, totalizando, assim, 81 pontos não ajustados.

Tabela 6.7 - Casos de Uso do Sistema de Secretaria de Câmaras Municipais

CASOS DE USO	COMPLEXIDADE			
	Simples (5)	**Médio (10)**	**Complexo (15)**	**Pontos**
Cadastrar Proposituras			X	15
Cadastrar Protocolo		X		10
Preencher Proposituras			X	15
Controlar usuários para acesso	X			5
Cadastrar Ofício			X	15
Encaminhar Ofício			X	15
TOTAL				**75**

A Tabela 6.8, contém os dados iniciais de planejamento e os valores finais do Sistema de Secretaria de Câmara Municipal. Nesse caso, em decorrência de dados históricos, considerou-se o NDE inicial com valor 5. Novamente, apenas um desenvolvedor participou desse projeto.

O Fator de Complexidade Técnica foi calculado em 1,06, pois apenas alguns dos fatores tiveram influência no sistema desenvolvido, como descrito na Tabela 6.9, e os demais fatores foram fixados em 3, que é o valor médio, que aproxima o FCT do valor 1. Na Tabela 6.9, além do Fator, Peso e Influência, também temos o motivo pelo qual o valor da influência foi calculado.

Tabela 6.8 - Valores Iniciais de Planejamento e Valores Finais do Sistema Secretaria de Câmaras Municipais

Descrição	Valor
Fator de Complexidade Técnica	1,06
Fator Ambiental	0,83
Pontos por Caso de Uso Ajustados	71,26
NDE Inicial (Dado Histórico da Empresa)	5 horas homem
Tempo Total Inicialmente Estimado	356,32 horas
Tempo Total Efetivamente Consumido	323,45 horas
Erro na Estimativa	10,15%

Tabela 6.9 - Cálculo do Fator de Complexidade Técnica.

FCT	Descrição	Peso	Influência	Fator	Motivo
F1	Sistema distribuído	2	4	8	O sistema projetado para ser distribuído.
F2	Tempo de Resposta	1	3	3	Valor médio (3)
F3	Eficiência	1	3	3	Valor médio (3)
F4	Processamento complexo	1	3	3	Valor médio (3)
F5	Código reusável	1	5	5	Sistema com componentes reutilizáveis
F6	Facilidade de instalação	0,5	3	1,5	Valor médio (3)
F7	Facilidade de uso	0,5	3	1,5	Valor médio (3)
F8	Portabilidade	2	3	6	Valor médio (3)
F9	Facilidade de mudança	1	3	3	Valor médio (3)
F10	Concorrência	1	3	3	Valor médio (3)
F11	Recursos de segurança	1	3	3	Valor médio (3)
F12	Acessível por terceiros	1	3	3	Valor médio (3)
F13	Requer treinamento especial	1	3	3	Valor médio (3)
FCT = 1,06					

O Fator Ambiental foi calculado em 0,83, analogamente ao Fator de Complexidade Técnica, em que alguns fatores tiveram influência e outros foram fixados no valor médio 3, como proposto por Karner [KARNER,1993]. Na Tabela 6.10, estão relacionados os fatores que contribuíram para o cálculo do Fator Ambiental.

Os Pontos por Caso de Uso ajustados totalizaram 71,26. Baseado neste resultado, calculou-se o tempo estimado para o desenvolvimento do módulo, multiplicando os Pontos por Caso de Uso (71,26) pelo NDE de valor 5, pois esse valor já foi apresentado pelo referido *desenvolvedor* em projetos anteriores. Dessa forma, a previsão para o desenvolvimento do sistema foi de 356,32 horas. Como o sistema foi desenvolvido em 323,45 horas, esta estimativa apresentou um erro de 10,15 % ou de 32,87 horas, que é considerado bom para um planejamento.

Tabela 6.10 - Cálculo do Fator Ambiental

FCT	Descrição	Peso	Influência	Fator	Motivo
E1	Familiaridade com o processo de desenvolvimento.	1,5	4	6	*Desenvolvedor* já desenvolveu outros sistemas na NBS.
E2	Desenvolvedores em meio expediente.	-1	3	-3	Valor médio (3)
E3	Capacidade em Análise	0,5	3	1,5	Valor médio (3)
E4	Experiência com a aplicação em desenvolvimento.	0,5	3	1,5	Valor médio (3)
E5	Experiência em Orientação a Objetos.	1	5	5	*Desenvolvedor* com experiência em Orientação a Objetos.
E6	Motivação	1	3	3	Valor médio (3)
E7	Dificuldade com a Linguagem de programação	-1	3	-3	Valor médio (3)
E8	Requisitos Estáveis	2	4	8	Sistema com baixa alteração de requisitos.
FA = 0,83					

Observa-se que, em comparação com os resultados do primeiro estudo de caso, o erro da estimativa foi quase 1/3 do erro da estimativa do primeiro. Isso se deve ao fato de que no Sistema Tapetes projetado na Linkway, a tecnologia utilizada foi um lançamento recente, que estava em amadurecimento. Foram utilizados os *frameworks* JSF 1.2 (*Java Server Faces*) e JPA 1.0 (*Java Persistence API*) poucos meses após a liberação para uso comercial. Enquanto que no Sistema Secretaria de Câmaras Municipais, apresentado neste estudo de caso, a tecnologia utilizada possui um alto grau de amadurecimento, pois está no mercado há diversos anos.

A interpretação dos dados apresentados na Tabela 6.11 é análoga àquela feita para a Tabela 6.5. Este estudo revela que as complexidades atribuídas para os Casos de Uso, de acordo com a técnica proposta por Karner, não mostraram nenhuma distorção, ou seja, os Casos de Uso complexos consumiram mais horas que os médios, que consumiram mais horas que os simples. Assim, o NDE decresceu à medida que o desenvolvedor se familiarizou com o módulo que estava desenvolvendo, ganhando mais experiência.

Outro fator de interesse nesta análise foi que, atribuiu-se um NDE inicial de 5, mas no primeiro Caso de Uso ele foi igual a 6.62. Assim, constata-se que houve

um erro no planejamento inicial de 115,30 horas a menos, que representa as 471,62 horas (estimativa após o primeiro ajuste), subtraída das 356,32 horas (estimativa baseada em dados históricos), resultando em 115,45 horas.

Tabela 6.11 - Resultado da Aplicação da Estratégia PCU|$_{PSP}$ na NBS

Caso de Uso	Etapa 1	Etapas 1 e 2		Etapa 4		Etapa 9	Etapa 2	Etapa 2
	Complexidade	Pontos por Caso de Uso ajustado		Horas Efetivamente Consumidas para conclusão do Caso de Uso		NDE	Estimado (horas)	Diferença em horas em relação ao planejamento inicial de 356,32 horas
		Individual	Acumulado	Individual	Acumulado			
1	Complexo	14,08	14,08	93,16	93,16	6,62	471,62	(-) 115,30
2	Médio	9,68	23,75	29,83	122,99	5,18	368,97	(-) 12,65
3	Complexo	14,08	37,83	60,58	183,57	4,85	345,79	10,52
4	Simples	5,28	43,11	13,41	196,98	4,57	325,62	30,70
5	Complexo	14,08	57,19	74,29	271,27	4,74	338,04	18,27
6	Complexo	14,08	71,26	52,18	323,45	4,54	323,45	32,87

Analogamente ao estudo de caso da Linkway, ajustou-se uma função aos valores do NDE de cada Caso de Uso da Tabela 6.11, resultando na função descrita na Tabela 6.12.

Tabela 6.12 - Função ajustada ao NDE

Variável	Valor
y0	4,60688 ± 0,06709
A1	6,75877 ± 1,34436
t1	0,82459 ± 0,13542
R^2	0,98973
Equação: y = A1 * exp(-x/t1) + y0	

Neste estudo de caso, o R^2, com valor 0,98973, demonstra uma melhor precisão do valor do ajuste em comparação com o primeiro estudo de caso, tendo em vista que se aproxima muito do valor 1. Com x sendo o número de Casos de Uso, A1, t1 e y0 constantes e y(x) o NDE, a Figura 6.2 representa um gráfico da equação y = A1 * exp(-x/t1) + y0, onde no eixo x têm-se os Casos de Uso e no eixo y o NDE.

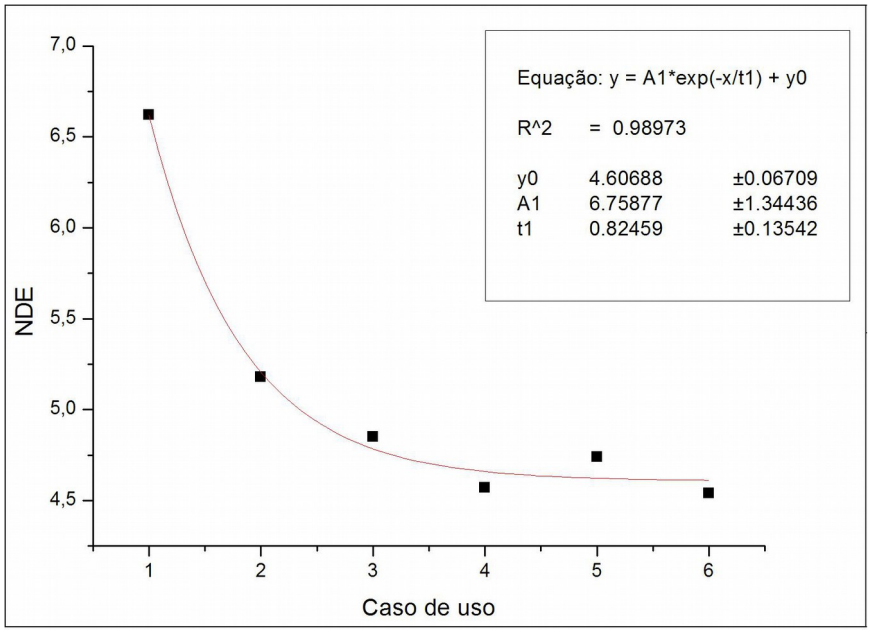


Figura 6.2 - Gráfico do NDE por Caso de Uso

Semelhantemente ao que foi exposto no estudo de caso da Linkway, novamente, esta função fornece indícios de que o NDE (4.60688 ± 0,06709) tende a uma constante dentro de um mesmo projeto, e essa constância dará a previsibilidade necessária para fazer o planejamento do *desenvolvedor* deste módulo. Desde que os Casos de Uso esteja bem estimados.

Em ambos os estudos de caso, o nível de influência dos Fatores Ambientais e de Complexidade Técnica iniciou com o valor três. Após a etapa 1 (Figura 5.1) obteve-se uma solução técnica mais apropriada para o sistema em questão, com base na qual os Fatores de Complexidade Técnica foram redefinidos.

Em nenhum dos dois sistemas houve alteração dessa solução. Nessa mesma etapa, os Fatores Ambientais foram acordados entre o *gerente* e o *desenvolvedor* e, apesar de serem revisados na etapa 8 (Figura 1), não houve alteração até o término do desenvolvimento. Isso ocorreu devido ao fato dos Desenvolvedores já terem trabalhado em outros sistemas nas respectivas empresas. Porém, novos Desenvolvedores

normalmente exigem um período maior de ajuste dos Fatores Ambientais.

6.5 Estudo de caso 3: A influência do PSP na gerência do projeto

Neste estudo de caso, a ênfase é mostrar os resultados da aplicação do processo PSP com a finalidade de auxiliar o planejamento e o acompanhamento de um projeto de software. A Estratégia $PCU|_{PSP}$, tem como um dos seus alicerces o PSP, pois em pequenas equipes de desenvolvimento de software, o desempenho individual é significativo com relação ao desempenho global do projeto. O processo PSP fornece aos desenvolvedores mecanismos que os ajudam a conhecer e melhorar a sua produtividade.

O processo de planejamento e acompanhamento de um projeto não é bem definido como a construção de um carro ou de um eletrodoméstico, como já foi mencionado nos Capítulos 3 e 4. Por exemplo, provavelmente a milésima geladeira da mesma marca e modelo, deve ser construída pelo mesmo processo que sua antecessora. Por outro lado, os processos utilizados na construção de um sistema de software nem sempre podem ser reutilizados integralmente na construção do próximo sistema.

As métricas utilizadas na construção de um sistema, nem sempre serão as mesmas da construção de outro sistema. Há casos em que um desenvolvedor consegue atingir a produtividade de 150 LOC/hora (*Lines Of Code/*hora) e, em outros casos, utilizando a mesma linguagem, não consegue ultrapassar a marca de 50 LOC/hora.

O fundamental não é definir a produtividade de um desenvolvedor em LOC/hora e rotulá-lo de acordo com seus recordes, mas dar condições para que ele seja capaz de medir sua produtividade constantemente, e de avaliar quais fatores que estão interferindo no seu desempenho. Por exemplo, um desenvolvedor deve perceber qual foi a interferência na sua produtividade quando utilizou um determinado *framework,* ou

quando muda de IDE (*Integrated Development Environment*).

Nas duas empresas abordadas neste trabalho a experiência mostrou que, normalmente, os desenvolvedores não conhecem seu desempenho. Eles não têm o hábito de avaliar sua produtividade, e nem de fazer estimativas de forma sistematizada, através de um processo que lhes permita trabalhar a melhoria contínua de suas estimativas, e o aumento de sua produtividade. Por outro lado, quando um *gerente* trabalha com uma equipe de Desenvolvedores que está capacitada em um sistema de melhoria contínua em seus processos pessoais de software, este *gerente* terá melhores condições de gerenciar o projeto de software como um todo, identificando precocemente os problemas que podem interferir nos prazos, quando estes são decorrentes de processos pessoais dos Desenvolvedores.

Os Desenvolvedores devem se comprometer com o planejamento pessoal e com seu auto-aperfeiçoamento, para que o *gerente* tenha melhores condições de gerenciar o projeto. Fazer o planejamento de um sistema inteiro, sem que os Desenvolvedores tenham a mínima noção de planejamento pessoal, aumenta os riscos de não se conseguir manter o projeto dentro dos custos e dos prazos estimados.

Por um lado os processos de software não são completamente definidos, pois a cada novo projeto, novas regras são utilizadas; por outro lado, os desenvolvedores desconhecem sua produtividade e suas limitações, dificultando o planejamento pessoal. Com a aplicação do processo PSP [HUMPHREY,2000], o *desenvolvedor* se compromete com sua produtividade, qualidade e planejamento e, conseqüentemente, ele estará se comprometendo com a qualidade e com o gerenciamento do projeto como um todo.

Através dos ciclos de melhoria contínua, proporcionados pela aplicação do processo PSP, os Desenvolvedores melhoram a capacidade de planejamento e,

portanto, desempenham melhor suas funções como profissionais dentro de uma organização. Eles exigem mais de si mesmos quando se avaliam em relação aos seus dados históricos, procurando sempre um melhor desempenho em relação aos seus resultados anteriores.

A experiência nessas empresas mostrou que o PSP deve ser implantado em uma equipe de *desenvolvedores* com a ajuda de um treinador. Os *desenvolvedores* que fizeram parte deste estudo de caso não tinham o habito de fazer o registro de tempo de suas atividades, e nem de fazer análises de seu desempenho após o serviço ser concluído. Percebeu-se que com a ajuda de um treinador os Desenvolvedores conseguiram manter a disciplina necessária para obter bons resultados em relação à capacidade de planejamento e produtividade. No entanto, houve desenvolvedores que não conseguiram manter a disciplina e, em função disso, prejudicaram o planejamento do projeto, como também o seu próprio crescimento profissional.

A ferramenta utilizada neste estudo de caso foi a *Process Dashboard*, comentada no Capítulo 2. Como o foco desta dissertação é o gerenciamento de projeto, a ferramenta foi utilizada para dar apoio à implantação do PSP até o nível 1.1, que tem foco no planejamento pessoal. Na Tabela 6.13 estão listadas as tarefas do PSP realizadas pelos Desenvolvedores neste estudo de caso.

Como o foco aqui está na forma de analisar os dados obtidos com a aplicação do PSP e também perceber a influência do PSP no planejamento do projeto, e na produtividade do *desenvolvedor*, serão explorados os dados somente de um *desenvolvedor*. A esse *desenvolvedor* foram atribuídas 37 rotinas. Lembra-se que os Casos de Uso são decompostos em cenários e estes em rotinas. Com o uso do PSP o *desenvolvedor* deve fazer algumas previsões para seu trabalho, as quais serão comentadas e analisadas em seguida.

Tabela 6.13 - Atividades do PSP realizadas no estudo de caso

Atividade do Processo PSP	Realização
PSP0	
Estimativa de Tempo Total	Sim
Tempo Efetivo por Fase	Sim
Tempo Acumulado por Fase	Sim
Tempo Acumulado por Fase (%)	Sim
Defeitos Injetados por Fase	Não
Defeitos Acumulados por Fase	Não
Defeitos Acumulados por Fase (%)	Não
Defeitos Removidos por Fase	Não
Defeitos Acumulados Removidos por Fase	Não
Defeitos Acumulados Removidos por Fase (%)	Não
Defeitos Removidos Após Desenvolvimento	Não
PSP0.1	
PIP	Sim
Padrão de Código	Sim
Estimativa de Tamanho Novo e Modificado	Sim
Tamanho Efetivo	Sim
Tamanho Acumulado Reusados	Não
Tamanho Acumulado Novos e Modificados	Sim
Tamanho Acumulado Total	Sim
Tamanho Acumulado Novo Reusável	Não
Estimativa de Tempo Por Fase	Não
PSP1	
Estimativa LOC/Hora	Sim
LOC/Hora Efetivo	Sim
LOC/Hora Acumulado	Sim
Estimativa de Tamanho (todos)	Sim
Relatório de Testes	Não
Estimativa de Novo e Modificado através de PROBE	Não
PSP1.1	
Índice de Custo de Performance	Não
% Reuso (*Plan, Actual, To Date*)	Não
% de Novo Código Reusável	Não
Planejamento de Tarefas	Sim
Cronograma de Tarefas	Sim

Estimativa do tamanho das rotinas em termos de LOC:

O *desenvolvedor* executou os passos pré-determinados pela Estratégia PCU|psp para cada rotina desenvolvida. Para montar o seu planejamento pessoal, ele estimou o tamanho de cada rotina em LOC. Observa-se na Figura 6.3, que após sete tentativas, o *desenvolvedor* atingiu uma significativa melhora em sua capacidade de fazer estimativa. As estimativas das rotinas 1, 11, 24 e 32 foram super valorizadas.

Nesses casos o *gerente* (*Scrum Master*) deve estar atento ao formulário PIP (*Process Improvement Proposals*) do *desenvolvedor*, uma vez que este deve conter uma explicação das bases do erro na estimativa e uma proposta sólida de melhoria.

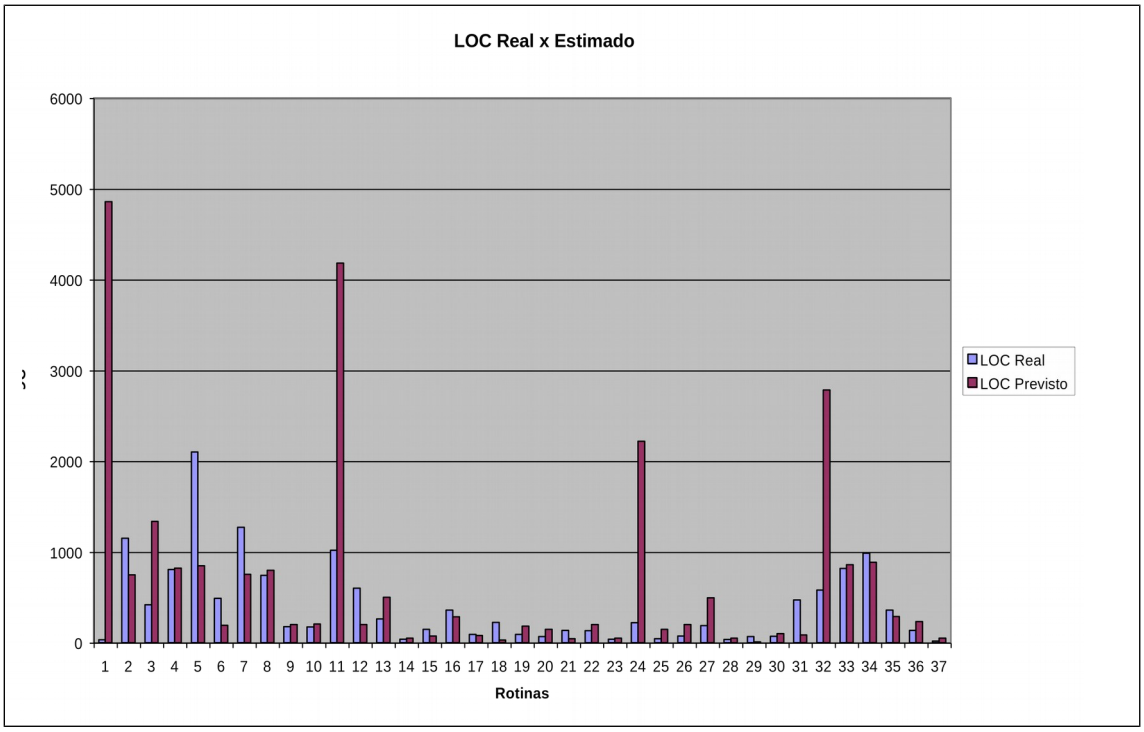


Figura 6.3 - LOC Real x Estimado

Na Reunião de Planejamento (Figura 5.1), é esperado um consenso entre *gerente* e *desenvolvedor*, principalmente em relação a custo e tempo de desenvolvimento dos Casos de Uso e das rotinas. Quando um *desenvolvedor* tem baixa capacidade de planejamento, esse consenso é prejudicado. O *gerente* deve ser capaz de analisar o desempenho do *desenvolvedor*, segundo os dados fornecidos pela aplicação do processo PSP, com o intuito de guiar as decisões na reunião de planejamento. Nessas reuniões, o *gerente* deve estar atento à capacidade de planejamento do *desenvolvedor* e a cada novo ciclo, estimular o *desenvolvedor* a melhorar sua capacidade de se auto gerenciar.

Estimativa do tempo para desenvolvimento das rotinas:

Na Figura 6.4, têm-se as estimativas de horas por rotinas, confrontadas com as horas realmente consumidas. Nota-se que, apesar do *desenvolvedor* em questão apresentar um razoável acerto no tempo de desenvolvimento de cada rotina, ele apresenta erros consideráveis em algumas delas, como por exemplo, nas rotinas 14, 16, 24 e 27. Nessas rotinas, ele superestimou o tempo, o *gerente*, nesses casos, deve mostrar ao *desenvolvedor* o tipo de erro que está sendo cometido e como fazer para melhorar sua a capacidade de estimar. Observou-se neste estudo de caso que, com os dados colhidos do PSP e representados no gráfico da Figura 6.4, tanto o *gerente* quanto o próprio *desenvolvedor* tem condições de avaliar, se as atividades realizadas para melhorar as estimativas atingiram o objetivo, ou se, pioraram a capacidade de acerto.

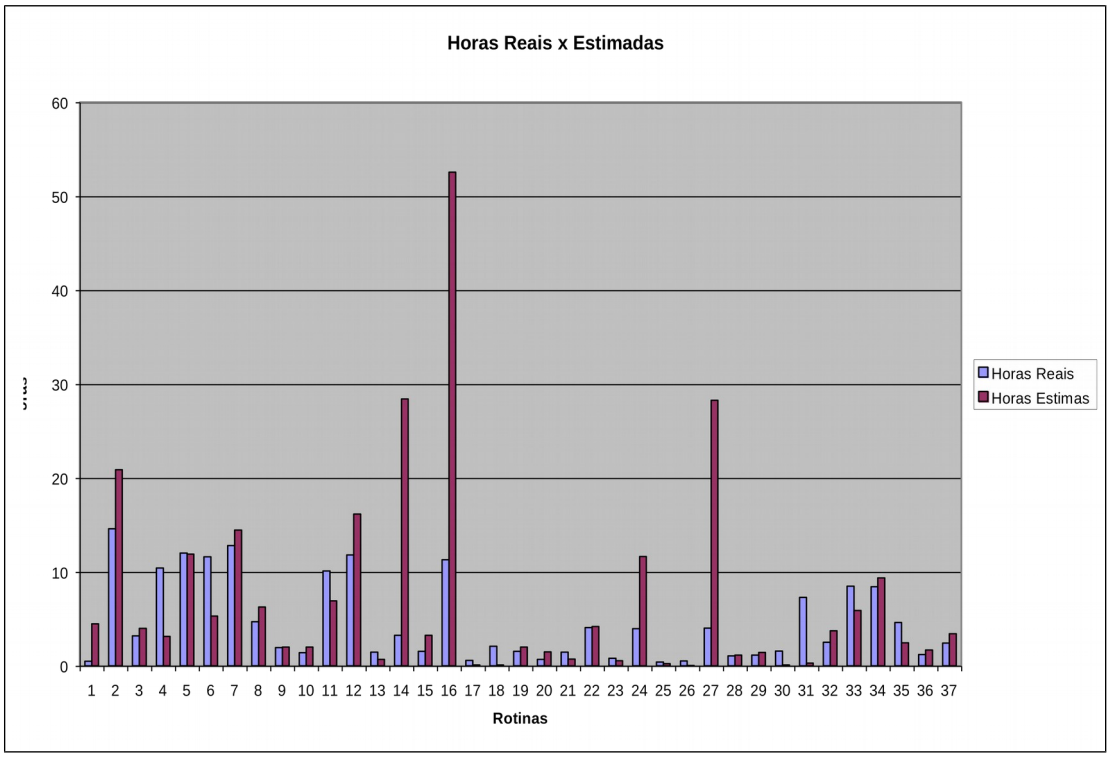


Figura 6.4 - Horas Reais x Horas Previstas

Avaliação da produtividade do *desenvolvedor*:

A produtividade do *desenvolvedor* é outra métrica que fornece subsídios para gerenciar o projeto. Na Figura 6.5 tem-se a produtividade média em LOC/hora para cada rotina. Observa-se que existe uma predominância de rotinas, em que o *desenvolvedor* trabalhou com a produtividade entre 50 e 150 LOC/hora. Em algumas rotinas ele ultrapassou a marca de 150 LOC/hora, e em nove rotinas ele ficou abaixo de 50 LOC/hora. Em todos os casos o *gerente*, em consenso com o *desenvolvedor*, deve entender o que influenciou a produtividade e justificar os motivos pelos quais a produtividade ficou longe de um valor médio. Neste estudo de caso, buscou-se justificativa quando a produtividade ficou abaixo de 50 LOC/hora, ou quando ficou acima de 150 LOC/hora. Se os fatores que interferem na produtividade são estudados e justificados, as próximas estimativas de tempo serão mais precisas. Por exemplo, na rotina 32, a produtividade atingiu um patamar de 232 LOC/hora, é improvável que um

desenvolvedor Java escreva código a essa velocidade, mas como ele estava usando um *Framework* de geração de linhas de código automática, essa produtividade foi possível.

Observou-se um consenso entre os Desenvolvedores tanto da NBS quanto da Linkway, de que o LOC/hora é influenciado por diversos fatores. Um dos fatores que o influencia é a utilização do padrão MVC (*Model View Control*), os Desenvolvedores tendem a uma maior LOC/hora quando estão trabalhando na camada *View,* e um menor LOC/hora quando trabalham na camada *Control*. Os Fatores Ambientais (Tabela 3.2), também interferem no LOC/hora, quando se tem um *desenvolvedor* com valores altos em "Dificuldade com a linguagem de programação", ou baixo em "Experiência em Orientação a Objetos", esses fatores reconhecidamente, diminuirão o LOC/hora. Analogamente, observou-se que os Fatores de Complexidade Técnica (Tabela 3.2) também interferem no LOC/hora. Sistemas que necessitam ser construídos com elevada preocupação com "Recursos de Segurança", ou então, estruturado para ser um "Sistema Distribuído", podem impactar no patamar de LOC/hora do *desenvolvedor*.

Rotular a produtividade do *desenvolvedor* em LOC/hora pode não ser uma boa prática de planejamento. Porém, ajustar o planejamento, conhecendo o valor médio em LOC/hora, e nos fatores que o influenciam, ajuda o *gerente* determinar melhor os prazos e custos do projeto. Observou-se que alguns Desenvolvedores ajustavam seu cronograma, baseando-se no seu valor médio de LOC/hora. Eles utilizavam as estimativas obtidas na aplicação da Estratégia $PCU|_{PSP}$ para ter uma previsão de quantas horas seriam necessárias para construção de um Caso de Uso, e tentavam estimar um LOC possível para esse Caso de Uso. Quando sua estimativa de LOC, levava a um LOC/hora muito acima, ou muito abaixo da própria média, ele buscava novos valores de LOC, que lhe projetariam uma taxa de LOC/hora mais

realista. Com esta técnica, o *desenvolvedor* conseguia um planejamento pessoal mais adequado à sua realidade.

Em relação à gerência do projeto, observou-se que a aplicação do PSP causa uma sensível melhora no acompanhamento. Isto se deu em função das variáveis que o *gerente* deve conhecer ao longo do Desenvolvimento, pois é necessário observar diariamente o desempenho da equipe em função da produtividade, capacidade de estimativas e planejamento pessoal. No método *Scrum*, a atividade de acompanhamento deve ser realizada nas reuniões diárias, em que o *Scrum Master* faz uma reunião rápida com a equipe, para saber como está o andamento do projeto [SCHWBER,2004]. Adotou-se, como uma forma de unir a Estratégia $PCU|_{PSP}$ com o método *Scrum*, a análise do PIP juntamente com a análise das questões pré-definidas nas reuniões diárias do *Scrum*. Esta forma de acompanhamento, permitiu ao *Scrum Master* o monitoramento do projeto de forma quantitativa e qualitativa.

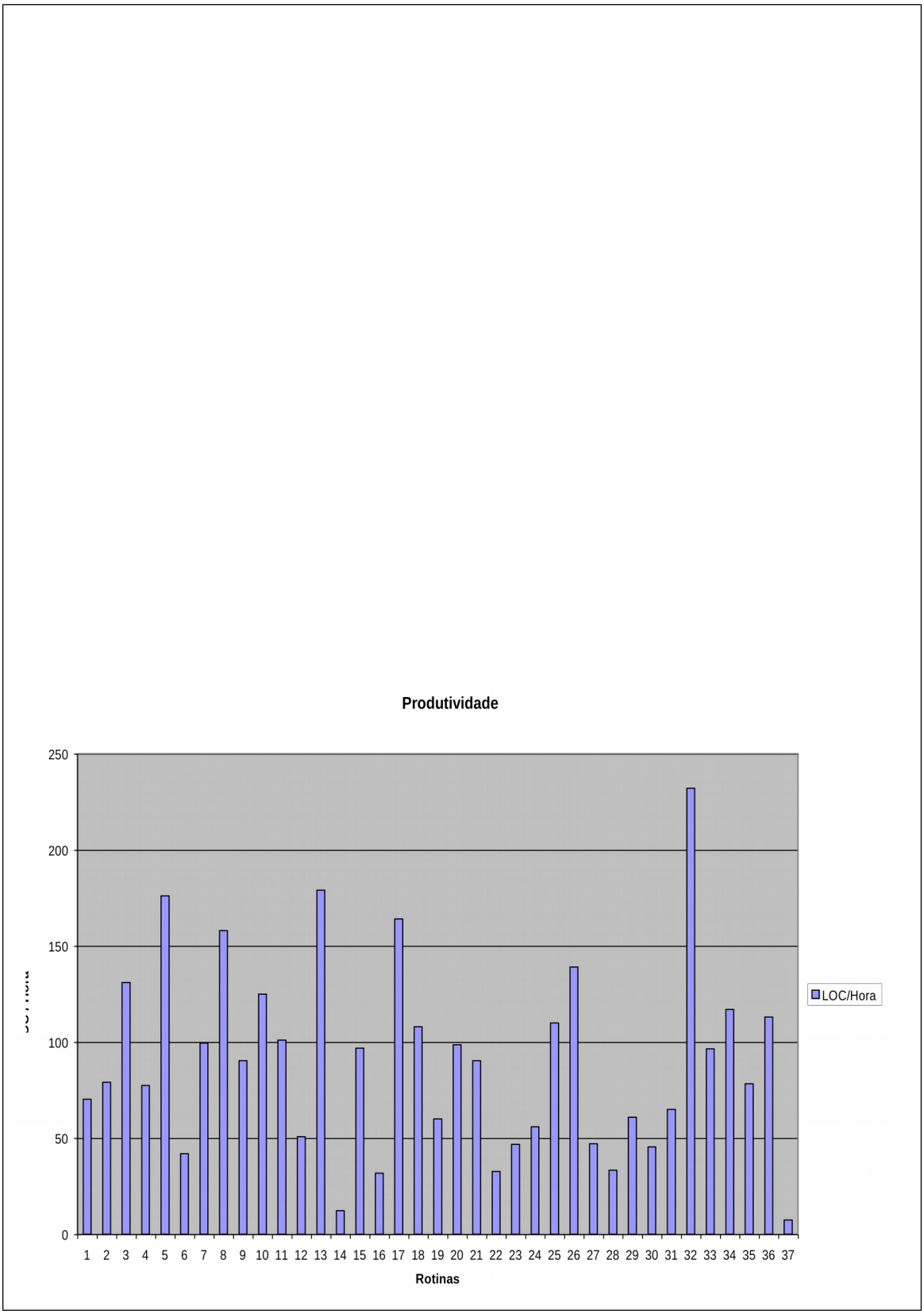


Figura 6.5 - Produtividade (LOC/hora)

Este estudo de caso evidenciou que o processo PSP, fornece ao *gerente* mecanismos, para que a gerência do projeto seja feita com informações mais realistas e

precisas. Como também, o *desenvolvedor* pode melhorar continuamente sua capacidade de planejamento, ajustando conforme o andamento do projeto, suas estimativas e sua produtividade.

6.6 Considerações finais

Neste capítulo foram apresentados três estudos de caso: dois deles exploraram a aplicação da estratégia PCU|$_{PSP}$ e outro explorou os ganhos obtidos do uso PSP no planejamento e gerência do desenvolvimento de software.

No primeiro estudo de caso explorou-se a aplicação da Estratégia PCU|$_{PSP}$ em uma empresa *Web*, com tecnologias em amadurecimento. Nesse estudo de caso observou-se que apesar das variações do NDE, que em alguns momentos, fez com que alguns Casos de Uso complexos, fossem realizados em menos tempo que Casos de Uso médios, foi possível tirar proveito do planejamento e do acompanhamento do projeto propiciado pela Estratégia, pois apesar das variações iniciais, após um período de desenvolvimento o *desenvolvedor* tendeu a uma constante, como visto na Figura 6.1, e os sucessivos reajustes do NDE (Tabela 6.5) permitiram um gerenciamento mais preciso do projeto.

No segundo estudo de caso explorou-se a aplicação da Estratégia em uma empresa cujo foco é o desenvolvimento de sistema para o ambiente *desktop*. Nesse estudo de caso, observou-se um maior acerto em relação à complexidade dos Casos de Uso e o tempo consumido para sua realização.

No terceiro estudo de caso explorou-se a aplicação do processo PSP como ferramenta de auxilio à gerência de projetos, bem como, uma importante ferramenta para a melhoria contínua dos Desenvolvedores.

Dessa forma, como pôde ser observado nos dois primeiros estudos de caso, a Estratégia PCU|$_{PSP}$ contribui de forma bastante positiva para o planejamento do

desenvolvimento de software, permitindo ao *gerente* planejar e reavaliar o planejamento de forma consistente através do cálculo do NDE, que o permite ter estimativa constantemente reajustada do progresso do projeto. Com isso ele tem também uma forma de avaliar o progresso dos Desenvolvedores ao longo dos projetos realizados na empresa de forma qualitativa e quantitativa.

Além disso, no terceiro estudo de caso fica evidenciada a vantagem de se usar o PSP em um ambiente de desenvolvimento, pois os Desenvolvedores além de melhorarem sua produtividade, e qualidade de desenvolvimento, também melhoram sua capacidade de planejamento pessoal, e conseqüentemente, ajudando no gerenciamento do projeto como um todo.

Salienta-se que, embora nesses estudos aqui apresentados só havia um *desenvolvedor,* a Estratégia pode ser aplicada em uma equipe. Nesse caso, para cada membro da equipe de desenvolvimento é aplicada a estratégia de forma individual. Cabe ao *gerente* fazer a composição dos trabalhos individuais, em um único projeto, como foi explicado no Capítulo 5. Recomenda-se, nesses casos, que o *gerente* promova a troca de informações sobre o andamento do projeto entre os membros da equipe para que as dependências sejam resolvidas de forma natural entre os Desenvolvedores.

Capítulo 7 – Conclusão

Apresentou-se neste trabalho a Estratégia PCU|$_{PSP}$ que auxilia a gerência de projetos, baseando-se no ajuste dos Pontos por Caso de Uso com dados provenientes da aplicação do PSP. Essa estratégia aborda tanto a etapa de planejamento, no que diz respeito ao tempo gasto para o desenvolvimento, como também o acompanhamento do progresso do projeto.

A Estratégia PCU|$_{PSP}$ foi extraída da experiência prática de duas pequenas empresas de software. Em ambas as empresas, buscou-se implantar o processo de "Gerência de Projetos", que consta nos modelos de qualidade de processo como o CMMI e o MPS.BR. Em função da implantação deste processo, houve a necessidade de dar suporte às atividades de planejamento e acompanhamento de projetos. Para dar suporte às atividades de planejamento, adotou-se o método PCU [KARNER,1993], com o objetivo de estimar o esforço em horas de desenvolvimento. E pela necessidade de ajustar as estimativas ao longo do projeto, adaptou-se o método PCU para que o esforço em horas fosse recalculado a cada Caso de Uso implementado. Outra adaptação no método PCU, em relação ao proposto por Karner, foi o cálculo dos Fatores Ambientais para cada *desenvolvedor*, ao invés de ser calculado para toda a equipe, permitindo assim, um controle maior do progresso dos Desenvolvedores. O processo PSP foi utilizado como ferramenta de melhoria contínua dos Desenvolvedores, e também, como suporte ao *gerente* nas atividades de acompanhamento do projeto de software.

A Estratégia confirma o fato relatado na literatura, mostrando que o Nível de Esforço do método de Pontos por Caso de Uso pode variar bastante da proposta original de Karner [KARNER,1993]. Nos estudos de caso aqui apresentados, o NDE variou próximo do dobro de uma empresa para outra, o que sugere que esse valor

deva mesmo ser ajustado para cada empresa, baseando-se também, em ajustes para cada *desenvolvedor*. Os dados históricos das empresas devem servir como valores iniciais, enquanto a contínua aplicação da estratégia é que fornecerá os valores corretos do NDE, ajustados para a realidade do projeto em questão. De fato, com o passar do tempo, é natural que o *desenvolvedor* se torne mais experiente e que isso interfira constantemente nos valores do NDE.

Tanto no estudo de caso da NBS quanto no da Linkway houve uma relativa estabilidade do NDE dentro do mesmo projeto, o que permitiu o constante acompanhamento do desenvolvimento através da análise dessa variável. Esse acompanhamento foi facilitado porque em ambos os projetos houve a participação de apenas um *desenvolvedor*, embora não seja provável que muitas dificuldades surjam se o número de desenvolvedores aumentar, uma vez que a diferença na aplicação da estratégia se resume na repetição do processo aqui descrito para cada um dos Desenvolvedores.

Uma vez ajustados os Fatores Ambientais e de Complexidade Técnica, a variação do NDE não se mostrou significativa. Caso aconteçam grandes variações do NDE, o *gerente* tem condições de avaliar objetivamente a razão do desvio ocorrido em relação ao Planejamento, e assim, tomar ações corretivas. Tanto a análise da variação do NDE e dos Fatores Ambientais, quanto à análise dos dados coletados pelo PSP fornecem ao *gerente* condições para um bom julgamento do desempenho dos desenvolvedores. Esse é um dos fatores diferenciais desta estratégia, pois a detecção de problemas no planejamento e no desenvolvimento é extremamente rápida e eficiente. Assim, o *gerente* pode assumir uma postura de Treinador, indicando diversas atividades para o aperfeiçoamento dos Desenvolvedores.

Ressalta-se que no contexto de empresas maiores, talvez a estratégia

tenha que ser adaptada pois, dependendo do tamanho da equipe de desenvolvimento, um controle individual não é factível. Além disso, nos dois estudos apresentados, houve a participação de um único *desenvolvedor*. Essa é uma limitação desses estudos, porém a Estratégia visa pequenas empresas e se houvesse mais desenvolvedores participando de um mesmo projeto, esse fato não causaria sobrecarga relevante na atividade de controle.

Foi possível a união da Estratégia PCU|$_{PSP}$ com o método *Scrum*, de forma bastante natural. O papel do *Scrum Master* corresponde na Estratégia ao papel desempenhado pelo *gerente*, e os mesmos princípios que regem o *Scrum Master* regem as atividades do *gerente*. Da mesma forma, o *Team* do *Scrum*, corresponde aos Desenvolvedores na Estratégia, com atribuições semelhantes. As atividades realizadas na reunião de Planejamento da Estratégia PCU|$_{PSP}$, fornecem subsídios para o Planejamento 1 e 2 do *Sprint*, facilitando tanto o dimensionamento do trabalho de um *Sprint*, como também, o planejamento individual de cada *desenvolvedor* (*Team*). Da mesma forma, as atividades da reunião de Controle da Estratégia PCU|$_{PSP}$ casam perfeitamente com o monitoramento previsto nas reuniões diárias do *Scrum*, permitindo ao *Scrum Master* (*gerente*) ter mais clareza do andamento do projeto e das dificuldades encontradas pela equipe, mas também, com o processo PIP (*Process Improvement Proposals*), realizado ao final de cada rotina, obter o compromisso da equipe (*Team*) com seu progresso e seu auto gerenciamento.

A união da Estratégia PCU|$_{PSP}$ com o método *Scrum* forneceu subsídios para facilitar a implantação do processo “Gerencia de Projetos” do nível G do MPS.BR, na maioria dos resultados esperados por esse modelo de qualidade. Da mesma forma, a Estratégia também contribui de forma simplificada, na implantação de outros processos do MPS.BR. A “Gerência de Projetos” do Nível B é beneficiada com a gerência quantitativa do NDE. No nível C, a Estratégia dá suporte à “Gerência de Riscos”,

quando se refere aos problemas encontrados na equipe de desenvolvimento. No nível E, a Estratégia auxilia nos processos de “Gerência de Recursos Humanos”, com as constantes avaliações dos Desenvolvedores, e no processo de “Gerência de Projetos”, através da medição de diversos produtos de trabalho.

Portanto, de acordo com os estudos de caso apresentados, foram obtidos indícios de que a Estratégia proposta nesta dissertação dá suporte ao planejamento e ao controle do projeto, permitindo um auto-aperfeiçoamento tanto do *gerente* quanto do *desenvolvedor*, tornando o desenvolvimento de sistemas mais preciso em relação a prazos e custos.

7.1 – Contribuições do Trabalho

A Estratégia PCU|$_{PSP}$ contribuiu nas atividades de gerência de projetos, dando suporte ao planejamento e ao acompanhamento do progresso dos projetos para pequenas empresas de desenvolvimento de software. Abaixo, estão listadas as principais contribuições do trabalho desta dissertação:

1. Na proposta original de Karner, o PCU é um método usado apenas para o planejamento do sistema como um todo. Nesta dissertação, o método é usado para planejar e acompanhar o desenvolvimento do projeto, tendo em vista que o gerente replanejará o sistema a cada Caso de Uso implementado. Além disso, com os ajustes efetuados a cada replanejamento, possibilita-se uma análise quantitativa do progresso da construção do sistema,
2. Na proposta original de Karner, os fatores ambientais eram avaliados no planejamento para a equipe como um todo, enquanto que na Estratégia PCU|$_{PSP}$ ele é avaliado para cada *desenvolvedor*. Os Fatores Ambientais se tornam também uma ferramenta usada no planejamento e acompanhamento da produtividade de cada *desenvolvedor*, tendo em vista que a cada término de

um cenário, os Fatores Ambientais são analisados novamente. A análise desses fatores poderá contribuir para o replanejamento do cronograma do projeto, mas também como um instrumento para a análise quantitativa da evolução do *desenvolvedor*.

3. A integração entre o processo PSP e o método PCU, propiciou a união do planejamento pessoal do *desenvolvedor* ao planejamento do sistema como um todo. Para o *gerente*, possibilita o acompanhamento do progresso da construção do sistema de software. Para o *desenvolvedor*, aprimora a sua capacidade de fazer estimativa de tamanho e de tempo, melhorando o seu desempenho no planejamento pessoal.
4. Em relação ao método *Scrum,* o cálculo dos Pontos de Caso de Uso em horas e os constantes reajustes do nível de esforço embasam a seleção, realizada pelo *Scrum Master,* do trabalho que será realizado em um Sprint, como também, auxiliam no planejamento individual de cada *desenvolvedor* (*Team*). As atividades realizadas na reunião de Controle da Estratégia PCU|$_{PSP}$ permitem ao *Scrum Master* (*gerente*) ter mais clareza do andamento do projeto e das dificuldades encontradas pela equipe, devido aos reajustes realizados no planejamento previstos na Estratégia.
5. Em relação ao modelo de qualidade MPB.BR, a Estratégia aplicada conjuntamente com o método *Scrum*, forneceu subsídios para facilitar a implantação do processo “Gerencia de Projetos” do nível G do MPS.BR, na maioria dos resultados esperados. A Estratégia PCU|$_{PSP}$ também contribui de forma simplificada, na implantação de outros processos do MPS.BR.

7.2 – Lições aprendidas

O desenvolvimento deste trabalho proporcionou um importante aprendizado, não só referente à extração da Estratégia PCU|$_{PSP}$ com o uso sistemático e conjunto do método PCU com o processo PSP, mas, principalmente, pela aplicação da Estratégia nas duas pequenas empresas. Um aprendizado importante é que deve ser ajustados à realidade da empresa, do projeto e dos *desenvolvedores* o valor do NDE, a granularidade dos Casos de Uso e a produtividade dos *desenvolvedores* em LOC/hora. Não sendo adequado buscar um valor padrão para ser utilizado em todos os projetos e em todas as organizações.

Outra importante lição é que, através da aplicação da Estratégia, é possível identificar as características de cada *desenvolvedor*, determinando seus pontos fortes e fracos. Uma vez que as características do *desenvolvedor* ficam mais expostas, o *gerente* tem que lidar de maneira pró-ativa para ajudar os *desenvolvedor*es a superarem suas dificuldades.

Também, observou-se que os erros no planejamento são rapidamente detectados, permitindo que ações corretivas sejam tomadas em tempo hábil para evitar problemas maiores. Outro aspecto interessante decorrente da aplicação da Estratégia é que, mesmo sem o *gerente* observar de perto o desenvolvimento das rotinas, os *desenvolvedores* cuidam, sozinhos, de manter o seu desempenho no mais alto patamar possível.

7.3 – Trabalhos Futuros

Como trabalhos futuros, pretende-se incluir nesta estratégia os seguintes objetos de estudo:

1. Neste trabalho, foi priorizada a aplicação do processo PSP até o nível 1.1.

Pretende-se, ajustar na Estratégia os outros níveis do PSP.

2. Aplicar a Estratégia em projetos com mais do que um *desenvolvedor*, com o objetivo de melhorar os controles do projeto para equipes maiores.
3. Explorar, também, outros tipos de métricas que poderiam melhorar ainda mais o planejamento.
4. Estudar mecanismos para fazer a estimativa de manutenção de software baseando-se no método PCU, com a finalidade de acrescentar na Estratégia de forma natural, ciclos de manutenção de software.
5. Relacionar à influência dos Fatores Ambientais com o Nível de Esforço de cada Caso de Uso, com o objetivo de compreender melhor as distorções encontradas nos estudos de caso. Como também, estudar outras variáveis que interferem nessa distorção, como por exemplo, à medida que o desenvolvedor se familiariza com o domínio da aplicação e com as técnicas utilizadas, ganhando experiência de uma maneira geral, seu Nível de Esforço pode decrescer.
6. Aprofundar o estudo dos benefícios da aplicação da Estratégia $PCU|_{PSP}$ em conjunto com o *Scrum*, na implantação do processo “Gerência de Projetos” do nível G do MPS.BR em relação aos atributos de processo AP 2.1.
7. Abranger mais fases do desenvolvimento.
8. Fazer um experimento controlado capaz de medir os benefícios do método de estimativa da Estratégia $PCU|_{PSP,}$ em relação às estimativas feitas de forma empírica por *desenvolvedores* experientes.

Referências Bibliográficas

[ALBRECHT,1979] Albrecht, A. J. (1979), Measuring application development productivity, Proceedings of SHARE/GUIDE IBM Application Development Symposium, pp. 83--92.

[BROGINI,2002] Brogini, G. G. (2002), *GOS-AE - Uma Abordagem Evolucionista para Garantia de Qualidade de Software*, Dissertação (Mestrado em Ciência da Computação e Matemática Computacional) - Universidade de São Paulo,São Carlos.

[CANDIDO,2004] Cândido, E. J. D. (2004), *Uma Simplificação da Técnica Análise de Pontos de Função para Estimar Tamanho de Aplicativos Web*, Dissertação (Mestrado em Computação) - Instituto de Ciências Matemáticas e de Computação, - Instituto de Ciências Matemáticas e de Computação. São Paulo, São Carlos.

[CAPILLA,2003] Capilla, R. & Duecas, J. C. (2003),Light-Weight Product-Lines for Evolution and Maintenance of Web Site, IEEE Computer Society, pp. 53--62.

[CHRISSIS,2003] Chrissis, M. S. S. (2003), *CMMI: Guidelines for Process Integration and Product Improvement*, Addison Wesley.

[COCKBURN,2002]Capability Maturity Model Integration Version 1.1. (CMMI-SE/SW, V1.2 – Continuos Representation), SEI Techinical Report CMU/SEI-2006-TR-001. Disponível em: http://www.sei.cmu.ed Cockburn, A. (2002), *Agile Software development*, Addison Wesley.

u/cmmi, Acesso em: 24 jun. 2007

[CMMI,2006]

[COHN,2005] Cohn, M. (2005), *Agile estimating and planning*, Prentice-Hall

[CTXML,2006] Programa CMM-3, *Disponível em: <http://www.ctxml.org.br/cmmi3/faq.aspx>. Acesso em: 13 mai. 2006.*

[DASHBOARD,2007] 'THE SOFTWARE PROCESS DASHBOARD PROJECT TEAM', *Disponível em: http://processdash.sourceforge.net. Acesso em: 02 mai. 2007.*

[DIAS,2005] Dias, M. V. B. (2007), *Um novo enfoque para o gerenciamento de projetos de desenvolvimento de software*, Dissertação de Mestrado, Universidade de São Paulo, Faculdade de economia, administração e contabilidade, Departamento de Administração, Programa de Pos-Graduação em Administração.

[HIGHMITH,2002] Highsmith, J.2002, (2002), *Agile Software Development Ecosystems*, Addison Wesley.

[HODGETTS,2004] Hodgetts, P. (2004), *Agile Planning, Tracking and Project Management Boot Cam*, XP Agile Universe Conference; Calgary, Alberta Canada; Disponível em: *http://www.agilelogic.com* - Acesso em 20 jul. 2007.

[HUMPHREY,1995] Humphrey, W. S. (1995), *A discipline for software engineering*, Addisonwesley, Pittsburgh.

[HUMPHREY,2000] Humphrey, W. S., (2000), *The Personal Software Process (PSP) ; CMU/SEI-2000-TR-022 ; ESC-TR-2000-022*.

[IPPD,2007] *CMMI and Integrated Product and Process Development (IPPD)*, Disponível em*: http://www.sei.cmu.edu/cmmi/ presentations/sepg01.presentations/ippd.pdf*, Acesso em*:* 10 jan. 2007.

[KARNER,1993] Karner, G. (1993), *Resource Estimation for Objectory Projects*.

[KNIBERG,2007] Kniberg, H. (2007), '*Scrum* and XP from the Trenches - How we do use *Scrum*', Disponível em*: http://infoq.com/minibooks/ scrum-xp-fromthe-trenches;* Acesso em*:* 20 nov. 2007.

[LARMARM,2004] Larman, C. (2004), *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Addison Wesley Professional, 3ª Ed.

[MANIFESTO,2001] *Manifesto for Agile Software Development*, Disponível em http://agilemanifesto.org. Acesso em: 05 nov. 2007.

[MARCHETTI,2000] Marchetti, P. A. S. (2000), *Avaliação de Pequenas Empresas segundo o Nível 2 do CMM*, Dissertação de Mestrado - Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos, São Carlos - SP.

[MARINESCU,2004] Marinescu, F. (2004), *Padrões de Projeto EJB*, Bookman, Porto Alegre.

[MPSBR,2007] *Melhoria de Processo do Software Brasileiro – Guia Geral (Versão 1.2).*, Associação para promoção da excelência do software brasileiro. Disponível em: http://www.softex.br/mpsbr, Acesso em: 20 out. 2007

[ORIGINPRO,2007] 'Data Analysis and Graphing Software', Disponível em: http://www.origenlab.com, Acesso em: 20 nov. 2007

[PAULK,1993] Paulk, M. e. a. (1993), *Capability Maturity Model for Software, Version 1.1. SEI Techinal Report CMU/SEI-93-TR-24*.

[PEREIRA,2007] Pereira, P. M. A. S. (2007), *Entendendo Scrum para Gerenciar Projetos de Forma Ágil*, Disponível em*: http://www.cesar.org.br/node/317;* Acesso em*:* 10 jan. 2007.

[PFLEEGER,2004] Pfleeger, L.Prentice-Hall, (2004), *Engenharia de Software – Teoria e Prática*.

[PRESSMAN,2006] Pressman, R. S. (2006), *Engenharia de Software*, McGraw-Hill, São Paulo.

[PROBASCO,2002] Probasco, L. *Function Points and Use Cases*, Disponível em: *http://www-128.ibm.com/developerworks/rational/library/ 2870.html,* Acesso em: 02 abr.2007

[ROCHA,2001] Rocha, J. C. W. K. C. (2001), *Qualidade de software*, Prentice Hall.

[SANTOS,2007] Santos, R. P. (2007), *Análise da Qualidade de Serviço de Empresas Avaliadas no CMM/CMMI*, Dissertação apresentada ao Programa de Pós-Graduação “Strictu Sensu” em Gestão do Conhecimento e Tecnologia da Informação, da Universidade Católica de Brasília, como requisito para obtenção do Título de Mestre em Gestão do Conhecimento e Tecnologia da Informação., Brasília.

[SCHWABER,2004] Schwaber, K. (2004), *Agile Project Management with Scrum*, Microsoft Press.

[SECM,2007] Systems Engineering Capability Model (SECM), Disponível em: *www.sei.cmu.edu/cmmi/publications/731-cmmi.pdf*, Acesso em: 10 out. 2007.

[SILVA,2004] Silva, L.F. (2004), *Estudo de alternativas para dar suporte à melhoria da qualidade de processo de software em empresas de pequeno porte com base no SW-CMM Nível 2*, Dissertação de Mestrado - Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos, São Carlos - SP.

[SOFTEX,2007] Associação para promoção da excelência do software brasileiro, Disponível em: *http://www.softex.br,* Acesso em: 30 nov. 2007.

[TAVARES,2000] Tavares, D. P. D. (2000), *Melhoria de Processo na Pequena Empresa: um estudo de caso*, Dissertação de Mestrado - Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos, São Carlos - SP.

[SATO,2007] Sato, D. (2007), *Uso eficaz de métricas em métodos ágies de desenvolvimento de software*, Dissertação apresentada ao Instituto de Matemática e Estatística da Universidade de São Paulo para obtenção do Grau de Mestre em Ciências, Área de concentração: Ciência da Computação.

[XAVIER,2005] Xavier, F. R. M. O. S. d. X. L. F. (2005), *Metodologia de Gerênciamento de Projetos - METHODWARE - Alinhado com os processos do PMBOK 3a edição*, Brasport.

[YOSHIMA,2007] Yoshima, R. (2007), *Entregue Software Funcionando – Gerenciamento de Projetos Ágil*, *in* 'Revista Mundo Java', pp. 10-19.