UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

IDENTIFICANDO DIFICULDADES E BENEFÍCIOS DO USO DO PSP APOIADO POR FERRAMENTAS DE 3ª. GERAÇÃO

RENAN POLO MONTEBELO

Orientadora: Profª. 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

Maio/2008

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M773id

Montebelo, Renan Polo.
Identificando dificuldades e benefícios do uso do PSP apoiado por ferramentas de 3ª. geração / Renan Polo Montebelo. -- São Carlos : UFSCar, 2011.
127 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2008.

1. Engenharia de software. 2. PSP. 3. SW-CMM - Modelo de capacidade de processo. 4. Qualidade de software. 5. Engenharia de software - experimentação. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Identificando Dificuldades e Benefícios do Uso do PSP Apoiado por Ferramentas de 3ª Geração"

RENAN PÓLO MONTEBELO

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 Sanches
(ICMC/USP)

Profa. Dra. Selma Shin Shimizu Melnikoff
(POLI/USP)

São Carlos
Maio/2008

Aos meus pais e ao meu irmão, dedico este trabalho.

AGRADECIMENTOS

À Prof[a]. Dr[a]. *Sandra Fabbri*, pela orientação, confiança, paciência, atenção e apoio nos momentos difíceis.

À minha *família* que, mesmo à distância, sempre me incentivou e acreditou em meu potencial.

Aos meus *amigos* de Araçatuba e de São Carlos, pelos ótimos anos de convivência.

Aos meus amigos do *Laboratório de Pesquisa em Engenharia de Software (LaPES)*, Arnaldo, Daniel, Dênis, Elis e Luiz, pelo companheirismo.

Aos professores *Ednaldo, Hélio, Jander, José Guimarães e Maria da Graça*, pela disponibilidade e auxílio na elaboração dos estudos de caso no Departamento de Computação da UFSCar.

Aos *alunos da graduação e desenvolvedores da Linkway*, pela colaboração nos estudos de caso.

Ao *CNPQ*, pelo apoio financeiro.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CASE	Computer Aided Software Engineering
CMM	Capability Maturiry Model
CMMI	Capability Maturity Model Integration
IDE	Integrated Development Environment
LOC	Lines of Code (Linhas de Código)
GQM	Goal-Question-Metric
ISEMA	In-Process Software Engineering Measurement and Analysis
PIP	Process Improvement Proposal
POO	Programação Orientada a Objetos
PSP	Personal Software Process
TSP	Team Software Process

RESUMO

Cenário: *O Personal Software Process™ (PSP) é uma metodologia que pode ser usada por desenvolvedores de software para melhorar a previsibilidade, a produtividade e a qualidade de seu trabalho pessoal. No entanto, a grande quantidade de atividades e métricas envolvidas no PSP torna a sua aplicação bastante trabalhosa e difícil. Várias ferramentas de apoio foram propostas com o intuito de facilitar a aplicação da metodologia, sendo que tais ferramentas evoluíram à medida que estudos sobre a aplicação do PSP eram realizados. Assim, existem hoje três gerações de ferramentas de apoio ao PSP, sendo que as ferramentas de 3ª geração caracterizam-se por coletar e analisar métricas de Engenharia de Software automaticamente, sem a necessidade de intervenção direta do desenvolvedor.* ***Objetivo:*** *O objetivo deste trabalho é determinar as dificuldades e os benefícios do uso de ferramentas de 3ª geração para o apoio à aplicação do PSP.* ***Método:*** *Utilizando o paradigma GQM, foram elaborados dois estudos de caso, sendo um em ambiente acadêmico e outro em uma pequena empresa de desenvolvimento de software. Ambos os estudos de caso envolveram a aplicação do PSP com o apoio de ferramentas de 3ª geração.* ***Resultados:*** *Os resultados mostram que ferramentas de 3ª geração apresentam algumas vantagens em relação às ferramentas de gerações anteriores, especialmente na fase de codificação. No entanto, tais ferramentas apóiam um número limitado de atividades do PSP, além de introduzirem novos problemas de adoção.* ***Conclusão:*** *Para que a aplicação do PSP tenha sucesso em longo prazo é necessário que a ferramenta de apoio seja completamente adaptada e integrada ao ambiente de trabalho do desenvolvedor, apresentando tanto características de ferramentas de 2ª quanto de 3ª gerações.*

ABSTRACT

Background: *The Personal Software Process™ (PSP) is a methodology that can be used by software developers to improve the predictability, the productivity and the quality of their personal work. However, the great number of activities and metrics involved in the PSP makes its application very laborious and difficult. Several PSP supporting tools have been proposed in order to make its application more feasible, and such tools evolved as more experiments were conducted with the PSP. There are now three generations of PSP supporting tools, whereas third generation supporting tools are characterized by being able to collect and analyze Software Engineering metrics automatically and unobtrusively.* ***Objective:*** *This study's objective is to determine the difficulties and benefits of using third generation tools to support the PSP application.* ***Method:*** *The GQM paradigm was used to set up two Case Studies, one being in the academic environment with undergraduate students and another in a small software development company. Both case studies featured the PSP application supported by third generation tools.* ***Results:*** *Results show that third generation supporting tools have some advantages over previous generations' tools, especially in the coding phase. However, third generation tools have limited PSP activities support and introduce new adoption problems.* ***Conclusion:*** *For a successful long-term PSP application, it is necessary that the supporting tools are completely adapted and integrated in the developers' work environment, with both second and third generations' tools characteristics.*

CAPÍTULO 1

INTRODUÇÃO

1.1. Contexto

O *Personal Software Process*^{TM} (PSP) [Humphrey, 1994; Humphrey, 1995b; Humphrey, 1997] é uma metodologia que desenvolvedores de software podem usar para melhorar a previsibilidade, a produtividade e principalmente a qualidade de seu trabalho. O objetivo do PSP é mostrar aos desenvolvedores de software que um processo bem definido e controlado pode ajudá-los a melhorar seu desempenho pessoal. O PSP reconhece que cada desenvolvedor tem características únicas e, assim, ao usar os resultados disponibilizados pelo PSP, os desenvolvedores podem determinar quais métodos são mais eficazes em seu próprio ambiente de trabalho.

Há relatos de uso do PSP que indicam ganho de desempenho de até 160% [Eman et. al., 1996], diminuição de defeits em até 530% [Verghese, 1996] e aumento da precisão de planejamento entre 32% e 67% [Humphrey, 1994]. No entanto, a aplicação do PSP é bastante trabalhosa e complexa. Um projeto desenvolvido utilizando o PSP pode exigir que o desenvolvedor preencha até 12 diferentes formulários que juntos possuem mais de 500 valores distintos [Johnson & Disney, 1998]. Essa sobrecarga de trabalho gerada pelas atividades do PSP foi um dos fatores responsáveis pela alta taxa de abandono da metodologia.

Para resolver tal problema, foram propostas várias ferramentas de apoio ao PSP para que a aplicação da metodologia se tornasse mais fácil e prática. As primeiras ferramentas foram apresentadas juntamente com a proposta original do PSP, e consistem em formulários de papel, *scripts, templates* e uso de cronômetro. Tais ferramentas, conhecidas agora como 1ª Geração, foram logo abandonadas, pois a coleta de métricas e os cálculos manuais eram extremamente trabalhosos e geralmente resultavam em problemas de qualidade dos dados [Johnson & Disney, 1998].

Na tentativa de diminuir o trabalho manual, ferramentas computacionais que realizam cálculos estatísticos, validação de dados e que constroem gráficos e tabelas foram elaboradas. No entanto, essas ferramentas - conhecidas como 2ª geração - ainda imputavam ao desenvolvedor a responsabilidade de fornecer os dados brutos. Ferramentas de 2ª geração tiveram sucesso em melhorar a qualidade das métricas coletadas, mas pouco modificaram o cenário de abandono do PSP pois os desenvolvedores ainda tinham que parar as suas atividades de desenvolvimento de software para dedicar-se a fornecer os dados do PSP para a ferramenta.

Ferramentas de 3ª geração surgiram com o propósito de tirar do desenvolvedor a responsabilidade de fornecer diretamente os dados do PSP para as ferramentas, pois as mesmas são capazes de coletar e analisar tais dados automaticamente e sem obstruir o trabalho do desenvolvedor. Desta maneira, o problema de abandono do PSP devido aos esforços de coleta e análise de dados é diretamente atacado. Essas ferramentas caracterizam-se por utilizarem "sensores" que são acoplados ao ambiente de desenvolvimento e, assim, capturam as atividades de desenvolvimento sendo executadas, tal como o tempo despendido modificando um arquivo-fonte. Porém, nem todas as atividades do PSP podem ser automatizadas, pois algumas métricas requerem o julgamento humano para serem coletadas, tal como número de linhas de código reusáveis. Assim, as características das ferramentas de 3ª geração dificultam que todas as atividades do PSP tenham suporte pela ferramenta.

1.2. Motivação e Objetivo

Considerando-se o contexto apresentado anteriormente, o objetivo deste trabalho é avaliar o impacto de ferramentas de 3ª geração na aplicação do PSP tanto em meio acadêmico quanto em empresas de desenvolvimento de software, identificando os pontos positivos e negativos dessa abordagem. Este trabalho também procura identificar abordagens e mecanismos que se mostrem facilitadores da aplicação do PSP, fazendo com que desenvolvedores utilizem a metodologia espontaneamente em seus projetos. Com esse fim, este trabalho propõe uma aproximação estreita entre a metodologia do PSP e ferramentas de apoio automatizadas e, neste contexto, é pioneiro.

A motivação para este trabalho é o fato do PSP se mostrar como uma excelente metodologia para melhorar a qualidade pessoal do trabalho dos desenvolvedores de software. Nesse sentido, o PSP é muito interessante para pequenas e médias empresas

que buscam aumentar a qualidade de seus projetos e desenvolvê-los dentro de prazos e custos estabelecidos, mas que nem sempre dispõem de recursos suficientes para implantar processos mais complexos. Ainda, o PSP pode inclusive acelerar a adoção posterior de processos mais complexos nas empresas [Ferguson, 1997].

O PSP também se mostra como uma ótima alternativa para ser aplicado em ambiente acadêmico. Atualmente, a maior parte dos alunos aprende a desenvolver software em disciplinas em que a única preocupação é adquirir conhecimento sobre algoritmos e linguagens de programação, e somente em disciplinas posteriores é que os alunos têm contato com conceitos de Engenharia de Software. Isso faz com que alguns alunos adquiram maus hábitos de programação que são difíceis de serem substituídos por boas práticas de desenvolvimento de software. O PSP, se aplicado desde as primeiras disciplinas de desenvolvimento de software, pode fazer com que os alunos aprendam a desenvolver software utilizando uma abordagem sistemática, disciplinada e mensurável que se preocupa com qualidade e prazos.

1.3. Organização do trabalho

Este trabalho está dividido em seis capítulos, sendo que neste apresentaram-se o contexto no qual o trabalho está inserido, os objetivos e a motivação para a elaboração do trabalho.

No Capítulo 2 apresentam-se os principais conceitos relacionados ao paradigma GQM, ao PSP e às ferramentas de apoio. Também são apresentados os trabalhos relacionados à aplicação do PSP, tanto em ambiente acadêmico quanto em empresas de desenvolvimento de software.

No Capítulo 3 é apresentado o planejamento de ambos os estudos de caso, seguindo o paradigma GQM.

No Capítulo 4 é apresentado o Estudo de Caso 1, realizado em ambiente acadêmico com alunos de graduação em Ciência da Computação da Universidade Federal de São Carlos.

No Capítulo 5 é apresentado o Estudo de Caso 2, realizado em uma pequena empresa de desenvolvimento de software da cidade de São Carlos.

As conclusões, lições aprendidas e trabalhos futuros são apresentados no Capítulo 6.

No apêndice A é apresentado o questionário utilizado em ambos os estudos de caso para caracterizar a população e para registrar a opinião dos estudantes e profissionais, anonimamente, sobre o uso geral do PSP. No apêndice B são apresentados os materiais elaborados para auxiliar os alunos na instalação, configuração e uso da ferramenta *Hackystat* e do ambiente de programação Eclipse.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

2.1. Considerações Iniciais

O objetivo deste capítulo é apresentar uma breve revisão bibliográfica acerca dos tópicos envolvidos nesta pesquisa.

Dado que o objetivo do trabalho é identificar os benefícios e as dificuldades do uso de ferramentas de apoio ao PSP de 3ª geração em ambiente acadêmico e em empresas, os tópicos essenciais do capítulo são a própria metodologia PSP e as ferramentas que a apóiam. Além disso, como o PSP está constantemente sendo explorado em diversos estudos, vários artigos relatando esses estudos foram identificados na literatura e são aqui apresentados.

Ainda, considerando a quantidade de dados que poderiam ser coletados e as diversas análises que poderiam ser realizadas, optou-se por modelar os estudos dessa dissertação de mestrado usando o paradigma GQM – Goal, Question Metric [Basili, 1992], com o objetivo de melhor direcionar o estudo e certificar-se de que tanto os dados a serem levantados quanto às questões que seriam interessantes responder estavam bem estabelecidas.

Assim, o capítulo está organizado da seguinte forma: o paradigma GQM, usado para planejar os Estudos de Caso é apresentado na Seção 2.2. A Seção 2.3 apresenta o *Personal Software Process* (PSP) e seus sete níveis de maturidade. As três gerações de ferramentas que apóiam a aplicação do PSP são apresentadas na Seção 2.4, com destaque para as ferramentas *Process Dashboard* e *Hackystat*, utilizadas no contexto deste trabalho. Os trabalhos relacionados, que incluem Estudos de Caso com ferramentas de 1ª e 2ª gerações, são apresentados na Seção 2.5. Por fim, a Seção 2.6 apresenta as considerações finais.

2.2. O paradigma Goal-Question-Metric

Métricas são importantes para avaliar processos e produtos de Engenharia, pois descrevem e relacionam medidas. Entende-se por medição o processo estabelecido para coletar medidas que possam caracterizar o objeto de estudo. Ao coletar e analisar medidas, é possível elaborar planejamentos, determinar pontos positivos e negativos de uma determinada abordagem e acompanhar o progresso de um projeto [Basili, 1992]. No entanto, elaborar um plano de medição, ou seja, quais métricas utilizar e de que maneira utilizá-las, não é uma tarefa trivial, sendo comum que se desperdicem recursos coletando e analisando medidas não relevantes, repetidas ou incompletas, enquanto medidas realmente relevantes podem ser ignoradas. Dessa maneira, a baixa qualidade das medidas coletadas pode fazer com que um trabalho tenha conclusões imprecisas.

Neste contexto, é importante que exista uma metodologia que efetivamente auxilie no processo de medição. O paradigma GQM é um mecanismo de medição para definir e avaliar um conjunto de objetivos [Basili, 1992] e foi proposto exatamente para minimizar as dificuldades inerentes à uma medição. O GQM tem uma abordagem sistemática *top-down* em que as medições são efetuadas com foco em objetivos e modelos [Basili, 1994]. Assim, devem-se deixar claro quais são os objetivos da entidade que está organizando a medição, assim como de seus projetos. Depois, devem-se definir quais são os dados que caracterizam os objetivos e então elaborar um modelo de interpretação que deixe claras as condições nas quais os objetivos são alcançados. Aplicar o GQM envolve os seguintes passos [Basili, 1992]:

1. Elaboração de objetivos da entidade organizadora, dos departamentos e de projetos;
2. Elaboração de questões e modelos que definam os objetivos da maneira mais precisa possível;
3. Especificação das medidas que necessitam ser coletadas para responder às questões elaboradas no passo 2;
4. Desenvolvimento de mecanismos de coleta de dados; e
5. Coleta, validação e análise dos dados em relação a um modelo de interpretação.

Uma única métrica pode estar relacionada a mais de uma questão, que por sua vez também pode estar associada a mais de um objetivo. Os objetivos, questões e

métricas fazem parte, respectivamente, dos níveis Conceitual, Operacional e Quantitativo do GQM [Basili, Caldiera, Rombach, 1994].

- **Nível Conceitual (*Goal*):** É no nível conceitual em que os objetivos são elaborados. Os objetivos podem ser relativos a Produtos, Processos ou Recursos, e são definidos de acordo com vários modelos de qualidade e de vários pontos de vista. Os objetivos podem ser Objetivos em Alto Nível (*Business Goals*), Objetivos de Software (*Software Goals*) e Objetivos de Medição (*Measurement Goals*).

- **Nível Operacional (*Question*):** Neste nível é elaborado um conjunto de questões que caracterizam os objetivos, que serão avaliados de acordo com um modelo de interpretação. As questões tentam caracterizar os objetos da medição (definidos no nível Conceitual), de acordo com determinados aspectos de qualidade.

- **Nível Quantitativo (*Metric*):** Um conjunto de métricas é associado a cada uma das questões, de modo a respondê-la de maneira quantitativa. Métricas são classificadas como objetivas se não há influência do ponto de vista na medição, como por exemplo, o tamanho de um programa em LOC. Por outro lado, métricas são subjetivas se dependem do ponto de vista, como o nível de satisfação dos clientes. Métricas subjetivas podem apresentar valores diferentes sob pontos de vista diferentes.

Na formalização dos objetivos (Passo 1), o GQM propõe o uso do *template* mostrado na Tabela 1. Dessa maneira, fica claro o contexto e o ponto de vista sob o qual o objetivo está sendo elaborado. Para fazer o relacionamento entre os Objetivos em Alto Nível com os Objetivos da Medição, foi usada a abordagem *GQM+Strategies* [Basili et. al., 2007].

Na formalização das questões, é importante que o objeto fique claramente caracterizado em relação aos objetivos e aos modelos utilizados, e que também fique claro como avaliar tais características [Basili, 1992]. As métricas propostas devem claramente responder às questões formuladas.

Tabela 1 – Template para determinação dos objetivos do GQM (Nível Conceitual) adaptado de [Basili et. al., 2007]

	Descrição	Exemplo de Aplicação
Objeto	Processo, produto, recursos	Analisar o **processo de teste**
Propósito	Caracterizar, avaliar, prever, motivar, melhorar	para **avaliar**
Foco	Custo, correções, confiabilidade, remoção de defeitos, ...	a **variação da densidade de defeitos de software**
Ponto de Vista	Usuário, cliente, gerente, desenvolvedor, corporação, ...	do ponto de vista dos **desenvolvedores de software**
Contexto	Fatores problemáticos, população, recursos, processos	no contexto de uma **fábrica de software para exportação**

2.3. Personal Software Process (PSP)

O PSP é um processo de melhoria contínua da qualidade de desenvolvimento de software, criado para ser aplicado individualmente pelos profissionais dessa área. Visto que grande parte do custo do desenvolvimento do software está associada aos esforços individuais, as habilidades e as práticas adotadas por esses profissionais influenciam amplamente os resultados obtidos. Torna-se imprescindível, portanto, a utilização de um processo de melhoria que não apenas diga o quê deve ser feito, mas que também procure informar como alcançar a qualidade individual desejada.

As práticas do PSP são derivadas do antigo CMM (Capability Maturity Model) [Paulk et. al., 1993] – atualmente CMMI [Chrissis et. al., 2004] - que fora desenvolvido pelo SEI (Software Engineering Institute). O CMM capturava as melhores práticas de desenvolvimento de software utilizadas por grandes corporações e era organizado em cinco níveis de maturidade. O PSP é o resultado de uma adaptação do CMM para o nível individual, e é dividido em sete níveis de maturidade (Figura 1). O PSP é constituído de formulários, relatórios, práticas e conceitos que progressivamente evoluem, tornando gradual o processo de melhoria. Assim, ao realizar as atividades propostas, o desenvolvedor consegue elevar o seu nível de maturidade dentro da metodologia PSP.

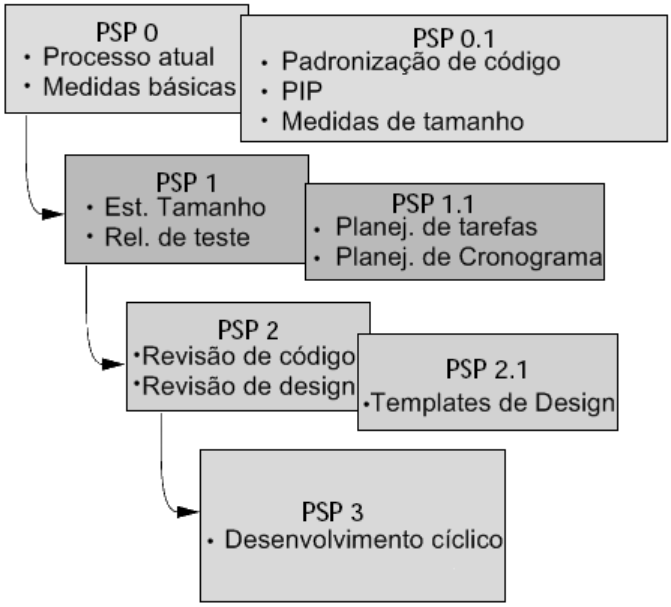


Figura 1 – Os sete níveis do PSP, adaptado de [Fergunson et. at., 1997]

No PSP, as atividades de desenvolvimento de software são divididas em Planejamento, Desenvolvimento (Projeto, Codificação e Testes) e Análise Final. Anterior ao planejamento assume-se que o levantamento de requisitos tenha sido realizado e esse artefato seja utilizado como ponto de início para as demais atividades.

Na fase de Planejamento, estimativas são realizadas e o desenvolvedor concentra-se em produzir valores que possam ser efetivamente alcançados. Na fase de Desenvolvimento, o profissional despende a maior parte do tempo efetivamente construindo o software, ou seja, seu objetivo é projetar, codificar e testar o software. Por último, como já diz o próprio nome, a fase de Análise Final é realizada para levantar informações a respeito de todo o trabalho desempenhado nas fases anteriores e, assim, estabelecer um comparativo entre que foi planejado e estimado com o que foi realmente executado.

O PSP se baseia fortemente em métricas, que são importantes pois podem identificar tanto aspectos positivos quanto aspectos negativos do processo de desenvolvimento de software. Coletar, analisar e acompanhar as mudanças em medidas é, em geral, um processo que demanda tempo, mas que se mostra compensador, principalmente para estudantes [Mengel, 1999]. Algumas das métricas importantes para o PSP são [Hilburn, 1999]:

- Tempo em cada uma das fases de desenvolvimento (planejamento, codificação, revisão, compilação, testes, análise final);
- Tamanho dos programas desenvolvidos (em número de linhas de código - LOC);
- Número de defeitos injetados e removidos em cada uma das fases de desenvolvimento;
- Número total de defeitos / mil linhas de código;
- Número de defeitos detectados na fase de teste / mil linhas de código;
- *Yield* – Número de defeitos encontrados antes da fase de compilação;
- *Appraisal Cost of Quality* (ACoQ)– percentagem do tempo dedicado à Revisão de Projeto e Revisão de Código;
- *Failure Cost of Quality* (FCoQ) – percentagem de tempo dedicado à compilação e testes;
- *Total Cost of Quality* – é a soma de ACoQ e FCoQ;
- *A/F Rate* - Relação entre ACoQ / FCoQ;
- Taxa de Revisão – número de linhas de código revisadas por hora;
- Taxa de remoção de defeitos – número de defeitos removidos por hora;

Essas medidas são importantes tanto para o desenvolvedor que está aplicando o PSP, quanto para o instrutor do curso de PSP que está avaliando e orientando os desenvolvedores [Hilburn, 1999].

As Seções 2.3.1 à 2.3.4 resumem as atividades do PSP de acordo com o nível de maturidade do desenvolvedor.

2.3.1. Base Pessoal (PSP Níveis 0 e 0.1)

O nível 0 representa apenas uma pequena introdução do PSP nas práticas de desenvolvimento do profissional. As atividades desse nível são bastante simples para que o desenvolvedor sinta pouca diferença entre o processo de desenvolvimento previamente em uso e os novos conceitos que estão sendo introduzidos. Algumas medidas começam a serem coletadas para que os alicerces do processo de melhoria sejam estabelecidos, como o tempo despendido e o número de defeitos identificados, por fase de desenvolvimento.

Mesmo que as estimativas e as avaliações mais complexas ainda não estejam presentes nesse nível, a coleta dos dados é extremamente importante para que os

cálculos estatísticos que serão realizados mais adiante sejam válidos e representem efetivamente a evolução do profissional desde o início da utilização do PSP.

O nível 0.1 acrescenta os conceitos de padronização de código e de tamanho, além de permitir que o desenvolvedor possa emitir propostas de melhoria para o seu processo pessoal (PIP – *Process Improvement Proposal*). Essa padronização do código faz com que a contagem de linhas de código tenha um valor consistente no decorrer do desenvolvimento dos programas, já que a inexistência desse padrão poderia gerar grandes distorções na contagem.

No nível 0.1 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 e então inicia a fase de desenvolvimento, onde são contadas as linhas de código geradas e o tempo despendido. Cada defeito identificado durante o processo é registrado e armazenado em um Relatório de Defeitos, para que se consiga identificar exatamente as maiores dificuldades do desenvolvedor.

2.3.2. Planejamento Pessoal (Níveis 1 e 1.1)

O nível 1 acrescenta atividades de planejamento ao PSP. Estimativas de tempo, juntamente com a elaboração do relatório de testes, são os principais acréscimos desse nível, que possibilita que desenvolvedores comecem a entender a relação do número de linhas de código produzidas em relação ao tempo despendido desenvolvendo o trabalho.

A partir do Nível 1, estimativas não são simplesmente calculadas livremente pelo desenvolvedor, pois se introduz o método PROBE [Humphrey, 1995b] para que o profissional tenha fundamentado uma maneira sistemática de estimar linhas de código. O método PROBE é um conjunto de técnicas estatísticas que foram especialmente selecionadas para o contexto do PSP, sendo a mais relevante destas a regressão linear. São acrescidos, portanto, um formulário para o armazenamento dessas estimativas e também um outro formulário voltado para a descrição e resultados dos testes a serem realizados no software. A introdução desse formulário de testes é importante para que o desenvolvedor comece a analisar a qualidade do software que produz.

O Nível 1.1 acrescenta a comparação do desempenho do desenvolvedor frente suas estimativas, juntamente com o formulário de planejamento de cronograma. A partir desse nível, o desenvolvedor realiza, na fase de Planejamento, as estimativas de quantidade de linhas de código do software a ser desenvolvido, além de uma estimativa de tempo mais detalhada em cada fase de desenvolvimento.

Essas novas atividades permitem ao desenvolvedor planejar mais detalhadamente as tarefas a serem realizadas de acordo com o seu tempo disponível, sendo possível acompanhar o trabalho e medi-lo freqüentemente ao longo do 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 identifiquem seus pontos fracos e fortes e, assim, aprimorar os aspectos que representam maiores problemas para o seu desempenho pessoal.

2.3.3. Qualidade Pessoal (Níveis 2 e 2.1)

A qualidade passa a ser o principal critério dos conceitos a serem acrescentados. O ponto mais importante desse nível é a introdução de *checklists* usados para a identificação precoce de defeitos. Inicialmente, esses *checklists* são oferecidos prontos pelo livro-texto, mas o desenvolvedor é encorajado a analisar seus pontos fracos no desenvolvimento e então personalizar tais *checklists*. O objetivo é que se identifiquem os defeitos introduzidos no projeto o mais rápido possível através de revisões formais. A Revisão de Código é a atividade mais importante desse nível.

Esse nível se preocupa não só com a qualidade da implementação, mas também com a qualidade do projeto do software. O PSP não diz exatamente como deve ser feita a modelagem ou a especificação do software, mas através de *templates* sugere como fazer especificações completas e consistentes.

No Nível 2.1 o objetivo é ajudar a reduzir os defeitos inseridos na fase de projeto, oferecendo critérios para a avaliação dos mesmos. A princípio, quatro propostas de modelagem são oferecidas pelo PSP, e os critérios de garantia de qualidade são inseridos no *checklist* de Revisão de Projeto. Assim, logo após o término da fase de projeto, o desenvolvedor usa o *checklist* para identificar e remover defeitos antes do início da fase de Desenvolvimento. Os modelos e *checklist* propostos são referentes à

interface com o usuário (operacional), interfaces de programação (funcional), modelo de estados e transições e modelo de lógica (pseudocódigo).

O PSP não obriga que os modelos oferecidos sejam utilizados para a especificação do software. O profissional pode utilizar o mesmo conceito desses modelos com qualquer outra modelagem apropriada. Como exemplo, ao utilizar diagramas UML pode-se acrescentar aos *checklists* de revisão os erros mais comuns introduzidos em diagramas UML.

2.3.4. Desenvolvimento Cíclico (Nível 3)

Até os níveis anteriores, o PSP é apropriado para programas com até algumas poucas mil linhas de código [Humphrey, 1997]. O Nível 3 possui o objetivo de ajudar o desenvolvedor na divisão de seu projeto de muitas mil linhas de código em projetos menores, adequados ao PSP níveis 2 e 2.1.

Para isso, o PSP propõe que uma porção principal do software (*kernel*) seja desenvolvida com a aplicação dos níveis 2 e 2.1, e que as diversas iterações necessárias para o desenvolvimento do restante do software sejam executadas sobre esse *kernel*. A cada ciclo de iteração, os critérios de qualidade discutidos nos Níveis 2 e 2.1 são aplicados, e assim garante-se que as próximas iterações possam ser realizadas sem que haja regressão no desenvolvimento.

O nível 3 propõe modificações um pouco mais profundas na estrutura do processo de desenvolvimento, como mostrado na Figura 2. Se antes se tinha um processo baseado em Planejamento, Desenvolvimento (projeto, codificação, compilação e testes) e Análise Final, agora o cenário de desenvolvimento é o seguinte:

- Planejamento Geral
- Projeto em alto nível
- Revisão de Projeto em alto nível
- Desenvolvimento Cíclico (Iterações compatíveis com PSP2.1)
    - Projeto detalhado
    - Revisão do projeto detalhado
    - Desenvolvimento de Testes
    - Revisão de Projeto

- Codificação
- Revisão de Código
- Compilação
- Testes
- Reavaliação e Reciclagem

- Análise final
- Integração

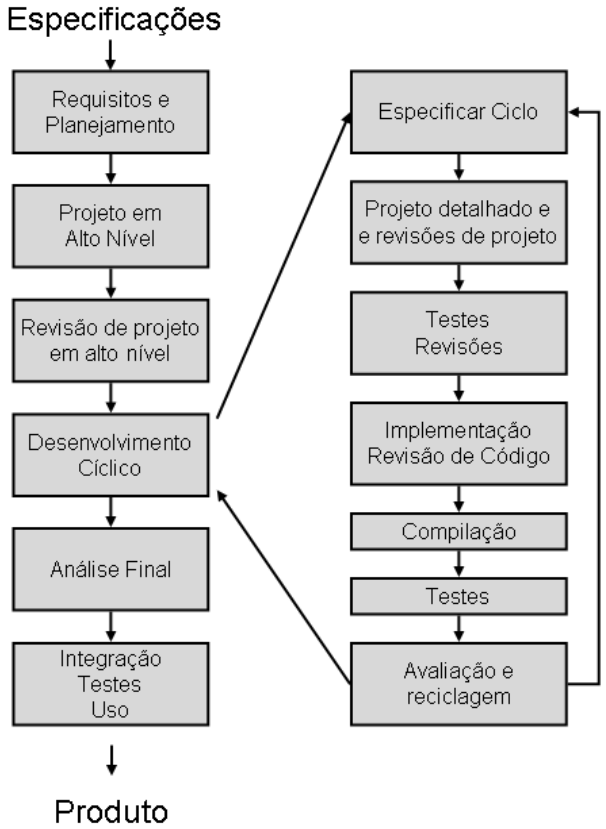


Figura 2 – Fases de desenvolvimento de Software no PSP Nível 3. Adaptado de [Humphrey, 1995]

Assim, nota-se que o processo é agora estabelecido em dois níveis distintos: o primeiro direcionado ao desenvolvimento do software em nível de abstração maior; e o segundo direcionado a cada ciclo a ser desenvolvido com a aplicação dos níveis 2 e 2.1 do PSP.

2.4. Ferramentas de Apoio ao PSP

Existem várias ferramentas de apoio disponíveis para o PSP. Essas ferramentas evoluíram conforme a aplicação do método também evoluía. Muitas propostas de ferramentas surgiram em resposta à identificação de problemas e dificuldades em se

aplicar o PSP, principalmente no meio acadêmico. As Seções 2.4.1 à 2.4.3 detalham as diferentes gerações de ferramentas.

2.4.1. Primeira Geração de Ferramentas de Apoio ao PSP

As primeiras ferramentas de apoio, ditas de primeira geração, foram propostas juntamente com a metodologia PSP por Watts Humphrey [Humphrey, 1995b] e consistem em formulários impressos de preenchimento manual. Devido ao grande número de cálculos e análises necessárias, esses formulários de uso manual eram difíceis de serem corretamente preenchidos e, portanto, desmotivavam o uso do PSP, pois o desenvolvedor precisava dedicar uma considerável parcela de seu tempo tratando manualmente os dados. Como relatado por alguns trabalhos relacionados [Johnson & Disney, 1998; Lisack, 1999], a falta de ferramentas computacionais é, ao lado de falta de motivação pessoal, o principal problema a ser contornado para a ampla aplicação do PSP.

Além da grande quantidade de tempo que o desenvolvedor tinha que dedicar à coleta e análise dos dados, ferramentas de 1ª geração também apresentaram um segundo problema. Como toda a coleta e análise de dados era realizada manualmente, era comum que os dados coletados apresentassem problemas de qualidade, afetando consideravelmente as análises elaboradas dentro do contexto do PSP [Johnson & Disney, 1998]. Desta maneira, ficou claro que outra abordagem deveria ser proposta para que a aplicação do PSP tivesse sucesso.

2.4.2. Segunda Geração de Ferramentas de Apoio ao PSP

Ferramentas computacionais surgiram para tentar solucionar os problemas identificados com ferramentas de 1ª geração. A princípio, foram propostas planilhas eletrônicas compatíveis com Excel, mas essas se mostraram muito pouco eficazes. Mais tarde, surgiram programas completos de apoio ao PSP, variando bastante quanto à tecnologia e metodologia empregadas.

Essas ferramentas computacionais, conhecidas como 2ª geração, trazem algumas facilidades tais como a construção automática de gráficos, a disponibilidade de uma base de dados históricos consistente e de fácil pesquisa, validação automática dos dados coletados, além de inúmeros cálculos realizados automaticamente. Essas ferramentas apoiavam diretamente o entendimento e a aplicação dos conceitos do PSP através de

exemplos e *templates* de aplicação. Tais ferramentas foram construídas em torno do PSP, dando apoio à todas as atividades da metodologia.

As ferramentas de segunda geração permitiram uma aceitação muito maior do PSP por parte dos alunos, pois os mesmos se viam amparados por um sistema que os livrava do trabalho considerado enfadonho e improdutivo, principalmente o preenchimento manual de tabelas. A disponibilidade de *templates* e de análises imediatas de dados permitem que as medidas coletadas tenham maior qualidade, tornando-as mais úteis.

Apesar do aumento da aceitação do PSP, o uso duradouro da metodologia pouco mudou. Estudos demonstraram que mesmo com a ajuda computacional para cálculos e preenchimento de tabelas, o desenvolvedor ainda assim precisava desempenhar dois papéis simultâneos: desenvolvedor de software e Engenheiro PSP, este último responsável por coletar e analisar métricas de Engenharia de Software pertencentes ao PSP. A constante troca entre os dois contextos fazia com que os desenvolvedores abandonassem o método assim que se viam em um ambiente onde o mesmo não era forçosamente aplicado.

Essa troca de contexto era considerada contra-produtiva pelos desenvolvedores, que se viam constantemente obrigados a interromper o desenvolvimento do software para abastecer a ferramenta com os dados do processo. Alguns alunos têm o hábito de modificarem apenas algumas linhas e então imediatamente compilam e testam o programa, tornado ainda mais difícil o controle correto do tempo pois as passagens pelas fases são muito rápidas. Além disso, cada defeito deveria ter seu tempo de remoção registrado, o que certamente introduz interrupções. Em alguns casos - como registrar o tempo de remoção de um defeito - era extremamente trabalhoso registrar as informações na ferramenta, de tal modo que o tempo necessário para inserir essa informação na ferramenta era maior do que o tempo necessário para efetivamente remover o defeito do software. Assim, era comum que os desenvolvedores deixassem de registrar algumas informações na ferramenta.

Um exemplo de ferramenta de 2ª geração é a Process Dashboard [Process Dashboard, 2008], qué é um projeto código-aberto que suporta todas as fases do PSP. A entrada de dados na ferramenta é manual, enquanto a análise de dados e a construção de

gráficos são parcialmente automatizadas. Um módulo contador de linhas de código-fonte semi-automático também é disponibilizado.

A ferramenta é desenvolvida 100% em Java e está disponível na Internet. Umas das principais características da ferramenta é a capacidade de personalizar totalmente o PSP, instanciando somente as atividades pertinentes à abordagem adotada. *Templates e Scripts* de uso do PSP são disponibilizados a cada etapa para auxiliar o desenvolvedor.

2.4.3. Terceira Geração de Ferramentas de Apoio ao PSP

De acordo com alguns pesquisadores, a aplicação efetiva e prolongada do PSP somente é possível se toda e qualquer coleta e análise de dados do PSP acontecer de maneira completamente transparente ao desenvolvedor [Johnson & Disney, 1998], o qual não seria obrigado a dedicar tempo extra para tais tarefas, eliminado assim a constante troca de contextos ("papéis"). De fato, em um cenário como este, o desenvolvedor não precisa mudar em nada a sua maneira de trabalho habitual, sendo que todo o tempo do desenvolvedor é dedicado integralmente ao planejamento, desenvolvimento e análise do software, com a impressão de que o tempo é mais bem aproveitado em tarefas consideradas úteis.

Assim, são propostas as ferramentas de terceira geração que se caracterizam por serem agentes de software inteligentes, geralmente acoplados aos ambientes de desenvolvimento integrado (IDE). Uma vez configurados, esses agentes podem coletar métricas de Engenharia de Software diretamente do ambiente de programação, sem a necessidade de intervenção direta do desenvolvedor. As métricas coletadas podem ser analisadas localmente ou enviadas a um servidor para análise.

No entanto, nem todas as atividades podem ser completamente automatizadas, pois existem atividades que necessitam, essencialmente, do julgamento humano para serem realizadas, como a identificação da fase de desenvolvimento em que um defeito foi inserido (Projeto ou Codificação?). Ferramentas de 3ª geração mudam, pois, a natureza de coleta de métricas e, portanto, não conseguem apoiar todas as atividades do PSP. Por este motivo, os conceitos do PSP acabam sendo menos expostos para o desenvolvedor se compararmos com ferramentas de gerações anteriores.

O fato das métricas serem coletadas automaticamente e, em alguns casos, enviadas para um servidor, faz com que ferramentas de 3ª geração tenham que lidar com problemas relacionados à privacidade dos dados. Se a ferramenta não garantir a privacidade dos dados, os desenvolvedores podem ficar receosos do aspecto de vigilância constante "Big Brother" [Johnson et. al., 2003] da ferramenta e então passam a evitar o seu uso, pois temem que seus dados pessoais, como produtividade, sejam utilizados para determinar remunerações de trabalho ou notas em uma disciplina.

A Tabela 2 compara as três gerações de ferramentas de apoio ao PSP. É possível observar que a geração 2 é um melhoramento direta da geração 1, mas que a geração 3 é, na verdade, um novo paradigma, que soluciona alguns problemas das gerações anteriores mas também insere novos problemas que antes não existiam.

Tabela 2 – Três gerações de ferramentas, adaptado de [Johnson et. al., 2003]

Característica	Geração 1	Geração 2	Geração 3
Trabalho de Coleta	Muito Alto	Alto	Nenhum
Trabalho de análise	Muito Alto	Médio	Nenhum
Troca de contextos	Sim	Sim	Não
Barreiras à adoção	Trabalhoso, troca de contextos	Troca de Contextos	Invasão de privacidade, baixa disponibilidade de sensores, suporte ao PSP incompleto
Conceitos do PSP	Muito expostos	Muito expostos	Pouco relacionados
Utilização	Abandonado	Muito utilizado	Em adoção

Como exemplo de ferramenta de 3ª geração pode-se destacar a *Hackystat*, que é uma ferramenta desenvolvida pelo Laboratório de Desenvolvimento Colaborativo de Software da Universidade do Havaí. Dos mesmos desenvolvedores da ferramenta LEAP [Johnson, 1999], esta é claramente uma tentativa de resolver o problema das constantes trocas de contexto necessárias para um desenvolvedor aplicar o PSP utilizando ferramentas de 1ª e 2ª gerações.

Esta ferramenta torna completamente automatizada as atividades de coleta e análise de dados. Sensores são acoplados às ferramentas de desenvolvimento, que então enviam informações para um servidor *Web* através do protocolo de alto nível SOAP [W3, 2007]. O desenvolvedor deve possuir uma conta pessoal no servidor e, por questões de segurança e privacidade, os dados somente são disponibilizados ao se informar o usuário e senha.

No servidor são efetuadas as análises dos dados que, então, disponibiliza os resultados através de consulta em uma página Web ou por emails diários. Se o usuário está temporariamente sem conexão com o servidor, um *cache* local é utilizado até que a conexão seja estabelecida. Os dados são enviados a cada 5 minutos, gerando vários *snapshots* de desenvolvimento. Para tratar possíveis problemas de privacidade dos dados, é possível configurar qual servidor receberá os dados dos sensores, que inclusive pode ser um servidor instalado na máquina local do desenvolvedor.

Ao contrário da ferramenta LEAP que possuía partes proprietárias e era um sistema monolítico, *Hackystat* é um sistema 100% código-aberto desenvolvido em Java, sendo, portanto, independente de plataforma. O servidor *Hackystat* possui mais de 15 KLOC (mil linhas de código); os sensores são, em geral, bem menores. A Tabela 3 resume as características da ferramenta.

Tabela 3 – Características da ferramenta Hackystat

Sensores disponíveis	Ant, BCML, Bugzilla, CCCC, Checkstyle, CLI, CppUnit, CVS, DependencyFinder, Eclipse, Emacs, Emma, FindBugs, Idea, JBlanket, Jira, JUnit, Jupiter, LoadTest, LOCC, Office, PMD, SCLC, SVN, Vim, VisualStudio
Tipos de dados	Atividade, Compilação, Defeitos, Commit de CVS, Cobertura, Dependências, Eventos, Métricas de arquivos, Desempenho, Atividade de Revisão, Testes Unitários
Tipos de análises	Métricas pessoais, Métricas de projeto, Métricas inter-projetos, Fluxogramas
Linguagens	Java, C#, C++, Ada, Chapel, Eiffel, Fortran, Lisp, Matlab, Pascal, Perl, Tcl, ZPL, XML, HTML, JSP, SQL
Relatórios e Gráficos	XML, HTML, CSV, Gráficos diversos

2.5. Trabalhos Relacionados

No meio acadêmico deve existir a preocupação de se educar uma nova geração de engenheiros de software preparando-os para amadurecer a indústria de software. Se o aluno desenvolve maus hábitos de programação, ele provavelmente carregará esses hábitos por toda a vida profissional. O PSP ensina aos alunos que o mais importante não é a tradução de um problema para uma linguagem computacional, mas sim entregar um produto dentro do prazo estipulado e sem defeitos, para que seja considerado um produto de qualidade. Embora existam muitos métodos de desenvolvimento de software, poucos são efetivamente usados porque engenheiros não são insistentes em

planejar, medir e acompanhar o seu trabalho. Assim, desconhecem seu real desempenho, talento, defeitos e habilidades.

Desde quando o PSP fora proposto, sua principal linha de pesquisa está relacionada a estudos e experiências que tentam elaborar melhores métodos e abordagens para introduzir a metodologia na sala de aula e na indústria. De fato, Morisio afirma que o mais importante para o sucesso não é a tecnologia ou o método em si, mas sim a sua aceitação [Morisio, 2000]. Embora o PSP seja um processo que pode ser explorado totalmente sozinho, foram relatadas experiências que mostraram que é mais eficiente constituir um ambiente de aprendizagem de PSP com várias pessoas [Humphrey, 1994]. Os resultados são melhores ainda quando os desenvolvedores têm experiência de programação, pois podem dedicar menos tempo aprendendo sintaxe de programação e mais tempo em melhorias de processo [Humphrey, 1994]. Por outro lado, desenvolvedores com pouca experiência em programação oferecem menor resistência ao método [Ceberio-Verghese, 1996], mostrando que não é trivial determinar qual é o nível ideal de experiência para uma ótima aplicação de PSP. No PSP, todos os defeitos são registrados, e não somente aqueles que chegam ao usuário final. Uma das principais métricas do PSP é a taxa de remoção de erros. Um dos usos dessa métrica foi mostrar aos desenvolvedores o quanto ineficaz é o tempo despendido em testes em relação à Revisão de Código (em termos de remoção de erros), através de um estudo de caso: em uma hora foi possível encontrar dez defeitos na fase de revisão de código, enquanto no mesmo código e no mesmo período de tempo foi possível encontrar apenas três defeitos na fase de testes [Humphrey, 1994]. No PSP, registra-se em qual fase o defeito foi injetado, em qual fase foi removido, o tipo de defeito e quanto tempo foi necessário para removê-lo. O desenvolvedor pode então construir um *checklist* de erros mais cometidos e assim evitar cometê-los novamente.

Para que o desenvolvedor desenvolva sistemas de alta qualidade é necessário que a taxa de injeção de defeitos seja minimizada e a taxa de remoção de defeitos seja maximizada. Há relatos em que a inserção de defeitos diminuiu em 530% durante a aplicação do PSP, aumentando a produtividade em 160% [Ceberio-Verghese, 1996]. É normal que haja aumento de número de defeitos encontrados durante o treinamento em PSP, pois esta situação está relacionada ao fato de que o PSP tem uma procura por defeitos mais minuciosa. É através de análises de dados de utilização do PSP como esses que algumas interessantes observações foram elaboradas [Zhong et. al., 2000]:

- Quando a média do número de fases de desenvolvimento de software envolvidas para remover um defeito diminui, há um aumento da taxa de remoção de defeitos e um aumento de linhas de código modificadas ou criadas por hora. Isso se deve ao fato do defeito estar sendo removido próximo do momento em que foi inserido. Se a média de fases afetadas é extremamente pequena, significa que os defeitos pertencem à atual ou anterior fase de desenvolvimento, e assim podem-se remover os defeitos com maior produtividade e menor custo;
- Baixa densidade de defeitos juntamente com um número pequeno de *yield* pode ser um indicativo de que a detecção de defeitos está ruim;
- *A/F Rate* mostra a relação entre as atividades de Revisão de Código e as atividades realizadas em Compilação e Testes. Quanto maior o valor desta relação, mais rapidamente estão sendo removidos os defeitos e, portanto, sob menor custo. No entanto uma taxa extremamente alta é contra-produtiva;
- Maior taxa de defeitos detectados por hora não necessariamente significa baixa qualidade do código, pois a detecção de defeitos é que pode estar se aprimorando;
- Valor baixo de *yield* geralmente indica Revisão de Código inapropriada;
- Se *A/F Rate* aumenta, linhas de código criadas ou modificadas por hora também tende a aumentar. Revisões de Código, portanto, afetam positivamente o número de linhas de código produzidas por hora;
- A média de fases de desenvolvimento envolvidas na correção de um defeito diminui quando *A/F Rate* aumenta. Portanto, o aumento de *A/F Rate* aumenta o número de defeitos removidos por hora e, conseqüentemente, também aumenta o número de linhas de código produzidas por hora;
- *A/F Rate* acima de 100% é associado a poucos defeitos de teste (o contrário também é verdadeiro). Portanto, esforços em Revisão de Código significam menos defeitos na fase de teste e, portanto, maior qualidade sob menor custo;

Um conceito errado é muito comum em torno do PSP e de processos de desenvolvimento em geral: processos limitam a criatividade dos desenvolvedores. Na verdade observa-se o contrário: através do PSP engenheiros evitam perder tempo com problemas triviais ou já enfrentados no passado, e assim podem dedicar mais tempo às atividades criativas e concepções de novas idéias.

Estudos mostram que engenheiros que somente lêem ou estudam o PSP acabam não aplicando o método. Por outro lado, engenheiros que fazem os exercícios durante o treinamento, em geral, utilizam efetivamente o PSP. Assim, após uma aplicação inicial em alunos da graduação, Watts Humphrey [Humphrey, 1995] concluiu que:

- Participação ativa é a chave do sucesso;
- Entre 150 e 200 horas são suficientes para completar os exercícios propostos. Com tempo disponível é possível completar todo o curso em 15 semanas;
- Se há recompensas, como notas por participação, quase todos os alunos terminam o curso dentro do cronograma proposto;
- Estudar sozinho não é tão eficaz, sendo que estudos em grupo são bem mais eficazes;
- Precisão de estimativas aumentou de 32% para 67%;
- Média de número de defeitos por mil linhas de código caiu de 132 para 51 defeitos/KLOC;
- Média de *Yield* (número de defeitos retirados antes da primeira compilação) caiu de 91 defeitos para somente 15;
- Média de defeitos em testes caiu de 33 para apenas 8;
- Produtividade praticamente se manteve a mesma, aumentando de 25 para 26 linhas de código por hora;
- É essencial que se cumpra todo o treinamento em PSP para que se tenha o efeito desejado;
- É interessante que o professor utilize os dados do PSP para efetivamente melhorar a sua aula, ao invés de contar somente com sua percepção para identificar os pontos em que pode melhorar.

O departamento de computação da *Enibry-Riddle Aeronautical University* implantou o PSP em seus dois primeiros cursos de programação (*CS1* e *CS2*). Os alunos da turma *CS1* focaram em gerenciamento de tempo, planejamento de produto, estimativa de tamanho de produto e geração de planejamento de produto. Os alunos da turma *CS2,* por sua vez, focaram em detecção de defeitos através de revisões formais de código, gerenciamento de defeitos e sua remoção do código. Esse programa chama-se *Doing Quality Work* e tem dois principais objetivos: trazer aos alunos experiências que os ajudem a entender a importância de um processo disciplinado e construir uma base para futuro desenvolvimento das técnicas do PSP e a sua aplicação em projetos

individuais ou coletivos. Os resultados, coletados ao final dos semestres, mostram que os alunos de ambos os cursos são capazes de praticar o PSP e que eles estão convencidos de que o PSP é útil para o aumento da qualidade dos trabalhos do curso [Towhidnejad & Hilburn, 1997]. No entanto, houve aspectos negativos da aplicação, especialmente quanto a exigir um grande esforço para coletar e analisar dados ao final do semestre, sem tempo hábil dos alunos sentirem os efeitos do trabalho dedicado ao PSP.

Fato semelhante aconteceu na *University of Utah*. Inicialmente os alunos são receptivos à idéia do PSP, sendo que 81% disseram que aquele material seria útil em suas vidas. Mas o PSP acaba sendo rejeitado pelo rigor exigido pelo método e constata-se que o ideal seria uma ferramenta de baixo ou nenhum impacto no processo de desenvolvimento que permitisse aos alunos concentrarem-se somente na programação [Williams, 1997]. Apesar disso, 88% dos alunos confirmaram que o tempo registrado foi o tempo efetivamente utilizado no desenvolvimento, mas o mesmo não aconteceu no registro de defeitos.

Nessa aplicação do PSP também se constatou que é apropriado "inverter" a ordem das atividades originalmente propostas pelo PSP. Foi mais eficaz ensinar aos alunos dos primeiros cursos de programação o gerenciamento de defeitos (PSP2), enquanto o gerenciamento de tempo (PSP1) foi ensinado aos alunos mais experientes [Williams, 1997]. Isso se deveu ao fato de que os alunos iniciantes já estavam sobrecarregados aprendendo programação, e atividades relacionadas ao PSP Nível 2 e 2.1 foram menos intrusivas. Também se constatou que é preciso tomar cuidado para que os ciclos do PSP3 não sejam rápidos demais, pois quando isto acontece os alunos são obrigados a dedicar mais tempo registrando as informações do que efetivamente desenvolvendo o software. Quando questionados ao final do treinamento, os alunos mostraram que obtiveram conhecimento sobre PSP e mostraram-se bastante interessados em dados sobre desempenho global da turma.

O *Software Engineering Institute* também publicou os resultados da aplicação do PSP em 298 engenheiros [Cannon, 1999]:

- Estimativas de esforço foram aprimoradas por um fator de 1,75;
- Estimativas de tamanho foram aprimoradas por um fator de 2,5;

- Qualidade do Produto (defeitos encontrados em testes) melhorou por um fator de 2,5;
- Qualidade do Processo (erros antes da compilação - *yield*) melhorou por um fator de 0,5;
- Produtividade se manteve a mesma;

Já na *Lund University* o PSP foi aplicado em alunos do quarto ano. Quando os alunos atingiram o Nível 2, no qual revisão de código é introduzida, os defeitos encontrados na compilação caíram de 58% para 35%. No entanto, a maioria dos defeitos detectados seria encontrada pelo compilador, o que não é ideal. Constatou-se, portanto, que se deve enfatizar a revisão de código para detectar erros que não são os de compilação [Wohlin & Wesslen, 1998].

Phillip Johnson e Anne Disney publicaram seus estudos de aplicação do PSP na *University of Hawai'i*. Seguindo a tendência de outros estudos, houve uma clara melhoria por parte dos alunos quanto às estimativas e quanto à noção da importância da Engenharia de Software. No entanto, um estudo de caso paralelo comprovou que o PSP, quando aplicado manualmente, injeta muitas distorções nos dados devido ao procedimento trabalhoso e delicado de coleta de dados [Johnson & Disney, 1998]. Apesar desse resultado, o estudo deixa claro que não se deve abandonar o PSP, mas deve-se evitar aplicar o processo manual do PSP, procurando sempre utilizar ferramentas que dão suporte ao PSP.

Apesar da grande quantidade de estudos que mostram o sucesso do PSP, deve-se atentar que os resultados basearam-se nos dados da própria aplicação do PSP [Johnson & Disney, 1998]. Deve-se tomar cuidado com este tipo de análise, pois tais dados podem ter sido distorcidos por causa de uma aplicação mal sucedida ou da falta de uso de ferramentas adequadas. Ainda, os dados podem ter sido distorcidos conscientemente pelos desenvolvedores para que atingissem desempenhos que teriam impacto positivo em salários ou notas de disciplinas, ou até mesmo distorcidos inconscientemente para que se reproduzissem os resultados esperados pelo PSP. Neste caso, o desempenho do processo é monitorado, mas é justamente o objetivo do processo melhorar o próprio desempenho. Este tipo de situação é de difícil análise, pois é muito fácil distorcer dados ou manipulá-los – consciente ou inconscientemente – para obter melhores resultados.

Para se avaliar o PSP corretamente, devem ser conduzidos estudos de caso com medidas externas não relacionadas ao PSP [Johnson & Disney, 1998].

Há um estudo em especial que põe em dúvida a eficácia do PSP. Lutz Prechelt e Bárbara Unger constatam que os dados positivos de uso do PSP advêm principalmente dos resultados do curso original de PSP contendo dez exercícios manuais [Prechelt & Unger, 2000]. Mas há dúvidas sobre a interpretação correta desses dados, uma vez que o processo vai mudando de um exercício para o outro, dificultando sua interpretação. Além disso, os exercícios de 4 a 10 são muito simples e com requisitos bem esclarecidos, o que faz com que os resultados pareçam otimistas demais. E o mais grave: não há controle, algo com que se possa comparar. Falta uma comparação direta entre uma turma treinada em PSP e outra não treinada. Os resultados da indústria são difíceis de serem interpretados por conta do complexo cenário e também pela falta de controle. Geralmente são comparadas as mesmas pessoas antes e depois do treinamento em PSP, ou então pessoas diferentes em projetos diferentes.

Susan Lisack, em seus estudos na *Purdue University* [Lisack, 1999], também constatou que há muitos problemas com os dados do PSP obtidos através de formulários manuais. Quando se submeteu um grupo de alunos ao treinamento PSP com formulários, somente 10 de 25 alunos submeteram formulários com dados utilizáveis. A maioria falhou em calcular o tempo total do projeto ou não subtraiu o tempo das interrupções. O número de linhas de código raramente foi preenchido e alguns alunos reportaram o tempo em horas ao invés de minutos, o que diminui a precisão dos dados. O número de defeitos reportados também foi, certamente, menor do que o número real de defeitos. Em um questionário ao final do treinamento, constatou-se que a maioria dos alunos não gostou do contato com o PSP e nem entenderam a sua importância, apesar de terem compreendido seus conceitos. Os alunos, em geral, também admitiram não terem dedicado tempo suficiente ao treinamento, enquanto metade admitiu que os dados fornecidos eram imprecisos. Como conclusão de seu estudo, Lisack coloca em dúvida a maturidade dos alunos do primeiro e segundo ano em seguirem um processo disciplinado como o PSP. Essa é uma informação que se considera muito importante no contexto desta pesquisa e que se pretende explorar em trabalhos futuros. Como já foi mencionado, no contexto deste estudo o objetivo não foi avaliar a efetividade do PSP propriamente dita, mas levantar um conjunto de informações que dêem subsídios a um

processo de implantação do PSP no qual os riscos e as dificuldades sejam previamente conhecidos, permitindo medidas de contingência e de precaução.

Em relação à maturidade dos alunos, deve-se sempre levar em consideração algumas diferenças importantes entre alunos calouros e alunos formados. Algumas dessas diferenças são listadas a seguir [Runeson, 2001]:

- Calouros escrevem programas menores na maioria das vezes. Provavelmente devido ao conhecimento limitado, produzem programas com menos funcionalidades;
- Calouros levam mais tempo para resolver um problema;
- Conhecimento sobre os vários níveis de PSP é o mesmo para calouros, formados e profissionais da indústria;
- Formados entregam os programas no prazo muito mais freqüentemente que calouros;
- Qualidade de dados de PSP dos calouros era inferior;
- Formados fazem mais perguntas sobre o processo de desenvolvimento durante as aulas; calouros fazem mais perguntas sobre linguagens de programação;
- A atitude dos calouros é mais positiva, pois são menos resistivos à idéia de que o PSP é importante. Formados questionam mais o processo e a necessidade de mudança;

Existem mais estudos que sustentam a tese de que o PSP é mais bem aproveitado por alunos mais experientes. Alunos iniciantes não estão preparados para aproveitarem completamente o PSP, e acabam tendo uma postura negativa em relação à metodologia devido ao tempo necessário para dedicarem-se às atividades do PSP [Maletic et. al., 2001]. Uma das alternativas neste caso é a simplificação do modelo, exigindo menos atividades. Ferramentas de apoio apropriadas ajudam muito, especialmente se forem de terceira geração. É positivo ter contato com o PSP o quanto antes, mas deve-se evitar o contato simultaneamente ao ensino de linguagens de programação.

Estudos como o de Susan Lisack reforçam a importância de um bom planejamento antes de aplicar o PSP. Como mostrado, as reações aos primeiros contatos com o PSP têm sido bastante intensas, tanto positivamente quanto de forma negativa.

Cabe ao instrutor analisar a abordagem proposta para evitar que os alunos tenham um contato traumatizante com o PSP.

O número de alunos em uma sala de aula parece ter pouco ou nenhum impacto sobre o uso do PSP. Durante a aplicação do PSP para uma sala de 360 alunos [Carrington et. al., 2001], os problemas enfrentados foram exatamente os mesmos encontrados quando se aplicou o PSP para cerca de 20 alunos, ou seja, reclamações quanto à quantidade de informações necessárias para coleta e a falta de ferramentas adequadas. Alguns alunos chegaram a afirmar que o PSP transformou as atividades de programação em atividades tediosas. Em geral, os alunos dizem que é fácil entender o PSP e que reconhecem sua utilidade, mas dizem ser muito difícil aplicá-lo, o que é um dado muito importante no contexto deste trabalho.

Um dos grandes diferenciais do uso do PSP é a disponibilidade de dados históricos que são usados como base para novas estimativas. Todos os trabalhos cujas propostas são semelhantes a esta dissertação aplicaram o PSP em um período curto, insuficiente para que os alunos acumulassem uma boa quantidade de dados históricos. A falta desses dados é uma constante dificuldade para uma aplicação bem sucedida do PSP.

Há dificuldade nos alunos em seguirem a disciplina exigida pelo PSP e em entenderem que as técnicas do PSP não são dogmas, mas sim um ponto de início de um processo pessoal. Nesse cenário em que poucos alunos continuam usando o PSP depois que o treinamento acaba, é possível questionar se o treinamento em PSP é mais eficaz do que um treinamento técnico, de acordo com os objetivos do PSP. Para isso foi feito um estudo [Prechelt & Unger, 2000] em que um grupo de alunos treinados em PSP se contrastou com alunos treinados tecnicamente na resolução de um mesmo problema, mas diferente dos problemas já apresentados durante o curso de computação.

Assim, o grupo P foi treinado em PSP seguindo os dez exercícios da proposta original do PSP. Já o grupo N foi treinado em Java através de cinco exercícios mais complexos. A experiência de programação dos dois grupos é similar. Apesar do treinamento, não se exigiu do grupo P o uso do PSP. Havia duas hipóteses: os programas desenvolvidos por alunos treinados em PSP seriam mais confiáveis e com

melhores estimativas; talvez o grupo P fosse um pouco mais produtivo do que o grupo N.

No entanto, também há alguns problemas de validade sobre a comparação aqui proposta: a turma P tinha acabado de ser treinada, então os resultados em longo prazo continuarão desconhecidos. Também seria mais apropriado propor um programa através de um problema real, e não um problema fictício como o proposto. Seria interessante também envolver profissionais da indústria para aumentar a contribuição do estudo. Os resultados são descritos a seguir.

A confiabilidade foi maior no grupo P (embora não claramente confirmada) porque programadores foram mais cuidadosos em checar erros ou situações inesperadas. O grupo P não estimou melhor do que o grupo N, mas o grupo treinado em PSP mostrou que conhece seus dados históricos, apenas não sabe utilizá-los a seu favor. O grupo N foi um pouco mais rápido, mas o grupo P construiu programas maiores, em geral mais elaborados. A produtividade foi, portanto, igual. O grupo P se mostrou mais homogêneo.

O grupo treinado em PSP estimou melhor o tempo necessário para remover cada defeito, além de comentarem mais o código e cometerem menos erros triviais. Muitos dos alunos do grupo P dizem se arrepender de não terem realizado Revisão de Código, sendo que uma parcela desse grupo afirmou que efetivamente usaria o PSP somente em projetos maiores. Por fim, apenas 25% dos alunos do grupo P usou efetivamente PSP, então é possível que a pesquisa reflita mais o grau de uso do PSP do que seus efeitos diretos.

São, principalmente, três os aspectos que influenciam no ensino do PSP: ambiente do curso, variações quanto à cobertura de níveis do PSP e ferramentas de apoio. O ambiente deve variar de acordo para quem o curso é voltado, de acordo com o nível do curso e com o conteúdo do assunto. O PSP tem se adaptado bem em quase todos os ambientes, mas há dificuldades em turmas de alunos iniciantes, ao mesmo tempo que o método depende muito das ferramentas de apoio.

Börstler et al. relatam algumas experiências de aplicação do PSP em várias universidades [Börstler et. al., 2002]. Na *Umea University,* o PSP foi integrado ao currículo de forma transparente. No caso, o PSP foi bastante modificado para que

ficasse leve o suficiente para ser aplicado em duas disciplinas de programação. O PSP teve seus conceitos expostos em uma palestra de 45 minutos e vários Estudos de Caso que comprovavam a eficácia do PSP foram apresentados. Os conceitos do PSP foram aplicados em três programas desenvolvidos pelos alunos. O uso do PSP era opcional, e apenas 6 de 78 alunos o utilizaram; os demais alunos relataram que optaram por não utilizar, por acharem que o PSP era um processo muito trabalhoso, e que este trabalho extra não valeria a pena. Nenhum dos 6 alunos que usaram o PSP relatou que melhoraram em relação aos aspectos esperados de uma aplicação do PSP.

Ainda segundo o trabalho de Börstler, A *University of Montana* obteve resultados satisfatórios somente quando o PSP foi aplicado integralmente e, apesar de uma inicial resistência, os alunos aprovaram o PSP. A *Drexel University* aplicou o PSP somente até o Nível 2, através de sete exercícios. Tal proposta ocorreu por falta de tempo hábil em aplicar o PSP em sua totalidade. Apesar de treinar os alunos de forma incompleta, alguns estudantes levaram os conceitos aprendidos para seus ambientes de trabalho.

A introdução do PSP na indústria se mostrou mais difícil, e os casos de sucesso geralmente são registrados quando há comprometimento dos gerentes com o PSP. De fato, como será visto no Capítulo 5 sobre o Estudo de Caso em realizado em uma empresa de desenvolvimento de software, o apoio do gerente foi fundamental para que todos os funcionários adotassem o PSP como rotina de trabalho.

Algumas empresas se destacam na aplicação do PSP, como a Motorola, que chegou a distribuir camisetas com o logotipo do PSP e realizar festas para o engenheiro que finalizava o treinamento, além de bonificações salariais [Ferguson, 1997]. A média de desenvolvedores que terminaram o curso de PSP é, em média, de 53% no ambiente acadêmico e de 43% na indústria; na Motorola, esse valor subiu para 77% [Eman et. al., 1996]. Esforços para uso do PSP na indústria incluem empresas como *Digital Equipment Corporation*, *Hewlett Packard* e *AIS*, esta que treina completamente seus engenheiros em PSP antes de alocá-los em projetos [Ferguson, 1997].

Engenheiros da indústria têm sido atraídos para o PSP por conta da constante pressão por resultados. Apesar deste cenário, uma parcela dos alunos que se treinaram em PSP na faculdade não estava utilizando o método na indústria, sendo que apenas

57% tiveram essa oportunidade [Eman et. al., 1996]. Na migração de PSP do ambiente acadêmico para ambiente industrial, as principais dificuldades são:

- Programas na indústria são inerentemente desenvolvidos em equipes;
- Falta de ferramentas adequadas;
- Falta de apoio dos gerentes;
- Falta de tempo extra para as tarefas do PSP;
- Cultura de desenvolvedores heróis;
- Resistência às mudanças;
- Falta de definição de um formato adequado de treinamento [Morisio, 2000];

O tempo de dedicação é um fator muito importante no treinamento do PSP na indústria. Dos engenheiros da *Union Switch & Signal* que precisavam concluir tarefas do PSP em horários fora do expediente, somente metade terminou o treinamento. A percentagem aumenta para 72% quando os engenheiros tiveram todo o dia para o treinamento, assemelhando-se ao treinamento aplicado em ambiente acadêmico [Ferguson, 1997]. Este comportamento é análogo ao comportamento dos alunos que se viram obrigados a dedicar parte de seu tempo de estudos de programação para o estudo do PSP [Lisack, 1999]. É importante, porém, que engenheiros da indústria encarem o treinamento em PSP como parte de seu trabalho regular e que os mesmos continuem a serem cobrados para que não revertam aos seus antigos hábitos de programação.

Verificou-se que com a aplicação do PSP, diminuiu-se a necessidade de atividades de gerência dentro das empresas, pois se desenvolve um comprometimento pessoal dos desenvolvedores em relação à qualidade [Ferguson, 1997] e há, portanto, um preparo natural da equipe de desenvolvimento para processos mais abrangentes como TSP e o CMM.

Um dos estudos bem sucedidos na indústria mostrou que aplicando o PSP foram encontrados 504 defeitos em um produto e, posteriormente, nenhum defeito foi encontrado por clientes [Hayes, 1998]. Muitas vezes, ao se usar o PSP para remover defeitos de um software, a curva de defeitos acaba se tornando semelhante à curva de defeitos ideal (Figura 3).

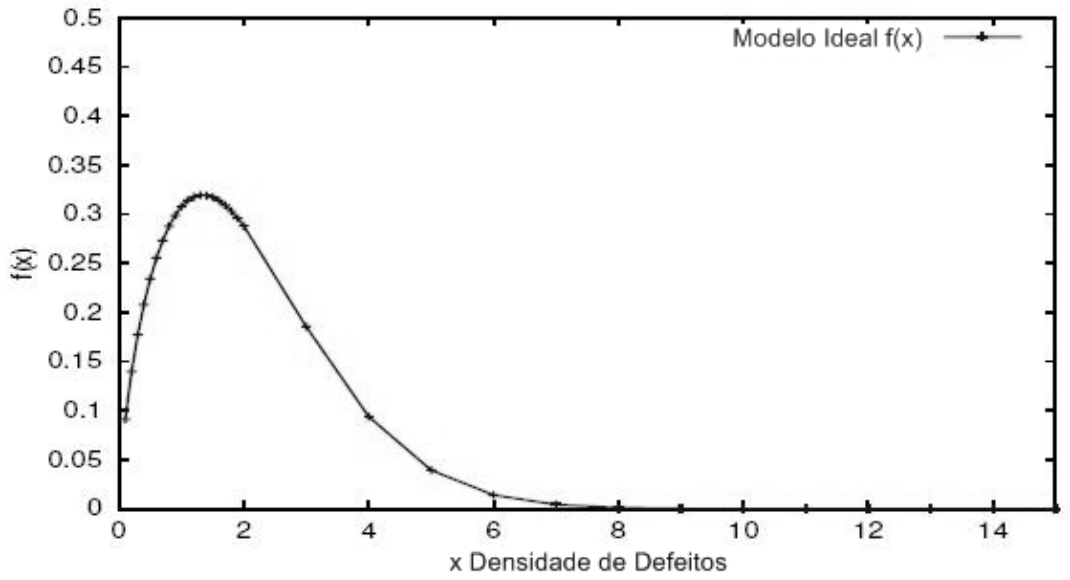


Figura 3 – Densidade de defeitos ideal. Adaptado de [Davila-Nicanor & Mejia-Alvarez, 2004]

A remoção de defeitos se mostrou bem distribuída em diversas fases do ciclo de vida de desenvolvimento do software, mostrando que ela não funciona somente nas fases de compilação e testes. Do total de defeitos, 41% foram injetados antes da codificação, 32% do total de defeitos foram removidos antes da codificação, tornando os dados de entrada de codificação praticamente sem erros. O método de Revisão de Código se mostrou muito eficaz, e praticamente dobrou de eficácia quando aplicado em dupla pelos engenheiros. O PSP, assim, se mostra um processo estável, pois os desenvolvedores efetuam mudanças pensando em melhorar o processo e, de fato, o processo melhora na mesma proporção esperada [Hayes, 1998]. Seguem mais alguns dados de aplicação de PSP na indústria [Cannon, 1999]:

- A empresa *AIS* reportou que o erro de estimativa de cronograma caiu de 394% para 10,4%; os defeitos por mil linhas de código caíram de 0,76 para 0,17;
- A *Motorola* envolveu o PSP em 18 projetos, sendo que somente um dos projetos apresentou problemas depois de lançado;
- A *Union Switch and Signal* também envolveu o PSP em 5 de seus projetos, sendo que nenhum apresentou problemas depois de lançado;

Um dos casos mais comuns de aplicação do PSP na indústria é a gerência da empresa autorizar que se adapte o PSP à empresa, mas não o contrário. Dessa maneira não são raras as diversas mutações do PSP na indústria. Por exemplo, há empresas que não geram estimativas para tarefas de manutenção por não justificar o tempo dedicado

[Morisio, 2000]. Há também uma adaptação do PSP para métodos formais [Babar & Potter, 2005].

Outro exemplo de modificação no PSP aconteceu durante a aplicação em uma empresa italiana considerada de tamanho médio [Escala & Morisio, 1999]. Toda atividade que consome tempo foi transformada em uma ação, que passa a ser a principal unidade de trabalho. Toda ação é executada em um documento, que inclusive pode ser o código-fonte. Ações se relacionam com defeitos: em uma ação podem ser encontrados zero ou mais defeitos, e da mesma forma zero ou mais defeitos podem ser removidos. Se a causa de uma ação é um defeito, classifica-se a ação como manutenção; caso contrário classifica-se como desenvolvimento. O planejamento passa a ser dividido em planejamento pessoal e planejamento de equipe - este chamado de projeto - e contém várias atividades, cada uma realizada por uma pessoa apenas e com tempo de realização entre duas ou dez semanas. O esforço é coletado individualmente e agregado ao esforço do time. Analogamente, os defeitos são coletados e analisados tanto no nível individual quanto no nível de equipe. O resultado dessa abordagem é que as métricas podem ser úteis em processos maiores, em especial o TSP.

A presença de uma ferramenta computacional também se faz necessária na indústria. Os engenheiros, sob pressão por resultados, podem adiar a coleta dos dados do PSP e tentarem reconstruí-los mais tarde, gerando inconsistências e distorções. Apesar das ferramentas não garantirem completamente a consistência dos dados, não há dúvida de que seu uso é essencial [Cannon, 1999; Johnson, 1998]. As ferramentas devem sempre tratar as questões de privacidade com seriedade, pois se não há esse cuidado os engenheiros podem forjar dados sob receio de não cumprirem certas metas e perderem seus empregos. O mesmo acontece com os alunos, que distorcem dados para não ficarem com notas baixas [Carrington et. al., 2001].

Há casos em que o PSP não é aplicado com sucesso. Maurizio Morisio [Morisio, 2000] constatou que o ambiente da indústria, por ser inerentemente um ambiente de equipe, dificulta o sucesso do PSP. Além disso, o uso de número de linhas de código como métrica de desempenho mostrou-se extremamente ineficiente, uma vez que atividades de reengenharia diminuíam o número de linhas de código, distorcendo a taxa de desempenho. Os engenheiros também abandonaram o registro de tempo de interrupções por considerarem esta uma atividade contra-produtiva.

As atividades de Desenvolvimento Cíclico geralmente ocorrem fora de ordem, muitas vezes confundindo os desenvolvedores que aplicam o PSP. Por esse motivo, alguns gerentes dizem ser o PSP um processo irrelevante e confuso, mas quando aplicado adequadamente e com disciplina, gerentes não têm esta opinião. Ainda é possível que um gerente combine dados de PSP de vários funcionários de forma a construir um modelo de estimativas que contenha custos e prazos para toda a equipe [Tomayko, 2003].

2.6. Considerações Finais

Neste capítulo foram apresentados os principais conceitos e trabalhos relacionados ao tema da dissertação. Assim, comentou-se sobre o paradigma *Goal-Question-Metric*, que foi utilizado no planejamento dos Estudos de Caso elaborados neste trabalho.

Também se comentou sobre os conceitos do *Personal Software Process*™ e seus setes níveis de maturidade. As características das três gerações de ferramentas que dão suporte ao PSP também foram apresentadas, em especial as ferramentas *Hackystat* e *Process Dashboard*, que fizeram parte dos Estudos de Caso.

Finalmente, comentaram-se vários trabalhos relacionados a cada um dos principais assuntos associados com o trabalho aqui proposto, isto é, a aplicação do PSP tanto em meio acadêmico quanto em meio industrial e uso das ferramentas de apoio de diferentes gerações. Como pode ser observado nos trabalhos comentados, existem várias vantagens em se usar o PSP, assim como também existem casos de insucesso. No entanto, acredita-se que as atividades propostas pelo PSP possam realmente ajudar o desenvolvedor no sentido de aumentar a qualidade do seu processo de desenvolvimento. Um ponto importante que foi observado é que houve relato de inserção do PSP em cursos de graduação de maneira transparente para os alunos, como se pretende fazer neste trabalho. Chama-se a atenção para o seguinte fato: os trabalhos mais recentes acabaram se distanciando do PSP propriamente dito e focando mais em ferramentas, especialmente de 3ª geração. Esta última é considerada uma ferramenta ISEMA (*In-process Software Engineering Measurement and Analysis*), que apesar de se propor a ser minimamente intrusiva, também acabou se distanciando do próprio processo do PSP. Isto se deveu ao fato de que muitas das atividades do PSP são muito difíceis ou até mesmo impossíveis de serem completamente automatizadas, e então tais ferramentas se

limitam a coletar e analisar métricas de Engenharia de Software de maneira generalizada, não se amarrando a um processo ou metodologia específica.

Os trabalhos apresentados neste capítulo foram muito relevantes e auxiliaram no planejamento e na elaboração dos Estudos de Caso presentes neste trabalho. O planejamento dos Estudos de Caso é apresentado no Capítulo 3, o Estudo de Caso em meio acadêmico é apresentado no Capítulo 4, e no Capítulo 5 apresenta-se o Estudo de Caso elaborado em uma pequena empresa de desenvolvimento de software.

CAPÍTULO 3

PLANEJAMENTO DOS ESTUDOS DE CASO USANDO GQM

3.1. Considerações Iniciais

Engenharia de Software é uma área multidisciplinar que inclui desde soluções técnicas até questões sociais e psicologia. Assim, deve ser tratada como uma ciência, implicando que métodos científicos devem ser utilizados para a tomada de decisões e para pesquisas. Nesse contexto, técnicas de experimentação são extremamente importantes para a Engenharia de Software, sendo que seu uso possibilita que se identifiquem e que se compreendam relações entre diferentes dados e variáveis [Wohlin et. al., 2000].

Estudo de caso é uma técnica de experimentação que é conduzida para investigar uma entidade ou fenômeno em um determinado período de tempo. Estudos de caso são estudos observacionais, diferentemente de experimentos, que são classificados como estudos controlados. Estudos de caso são apropriados para a avaliação de métodos de Engenharia de Software em ambientes industriais e para a avaliação de ferramentas, pois evitam problemas de escalabilidade que possivelmente existiriam em um experimento [Wohlin et. al., 2000].

Como em um estudo de caso pode haver muitas variáveis distintas, é importante que se utilize uma técnica que auxilie na coleta e análise dos dados. Caso contrário, é possível que esforços sejam desperdiçados na coleta de informações irrelevantes, incompletas, imprecisas e redundantes, resultando em conclusões equivocadas. Assim, o paradigma GQM foi utilizado para planejar a coleta dos dados dos estudos de caso presentes neste trabalho. Além de tratar da coleta de dados, o paradigma GQM auxilia a estabelecer um modelo de interpretação das métricas coletadas para que se defina, claramente, se os objetivos almejados foram atingidos. No entanto, estabelecer tal modelo com base em valores absolutos ou percentuais é extremamente complexo neste trabalho, pelos seguintes motivos:

- Falta de base histórica, uma vez que os estudos de caso apresentados neste trabalho são os primeiros a serem desenvolvidos, nesta área, no Departamento de Computação da Universidade Federal de São Carlos;
- Estudos de caso semelhantes relatados na literatura possuem muitas variáveis e configurações diferentes, que dificultam a comparação de resultados. Algumas das variações encontradas entre os diferentes estudos de caso são: número de participantes, duração, número de projetos desenvolvidos usando o PSP, uso de ferramentas de apoio de diferentes gerações, maturidade dos desenvolvedores, níveis do PSP aplicados durante o estudo de caso, entre outros;
- Os resultados da aplicação do PSP são essencialmente pessoais, de difícil generalização.

Assim, ao invés de aplicar um modelo de interpretação com valores absolutos ou relativos, como por exemplo, "queda no número de defeitos em 20%", procurou-se estabelecer um modelo de interpretação que compara o estado das métricas (*snapshot*) antes e depois da aplicação do estudo de caso, como por exemplo, "haver queda no número de defeitos encontrados".

Neste capítulo são descritos os planejamentos de ambos os estudos de caso presentes neste trabalho, nos quais foi utilizado o paradigma GQM, descrito em maiores detalhes na Seção 2.2. Na Seção 3.2 é apresentado brevemente o planejamento inicial, que foi elaborado antes do conhecimento dos problemas causados pela constante troca de contexto verificado em ferramentas de 2ª geração [Johnson et. al., 2003]. Na Seção 3.3 é apresentado o planejamento atual, referente aos estudos de caso elaborados tanto em meio acadêmico quanto em empresas de software. Na Seção 3.4 apresentam-se as considerações finais desse capítulo.

3.2. Planejamento Inicial

Um dos objetivos deste trabalho é estudar métodos e abordagens que auxiliem na implantação do PSP em disciplinas de programação do Departamento de Computação da Universidade Federal de São Carlos. Dessa maneira, os alunos poderiam desenvolver bons hábitos de programação desde o primeiro contato com algoritmos e linguagens de programação. Para atingir este objetivo, começou-se a construir a ferramenta PSP-TUTOR, uma ferramenta *Web* de 2ª geração, de apoio ao PSP.

A PSP-TUTOR caracterizava-se por ser realmente um Tutor de PSP para os alunos, em que estes só poderiam avançar para níveis mais altos de PSP uma vez que completavam todas as exigências e atividades do nível anterior. Conforme os alunos realizavam as atividades do PSP, os mesmos eram auxiliados pela ferramenta através de *templates* e *scripts* para que o preenchimento dos dados fosse corretamente executado. A intenção era que, gradualmente, os alunos fossem passando para níveis mais altos do PSP, cada um com sua própria linha de progresso pessoal.

Por ser uma ferramenta *Web*, os dados dos alunos sobre o uso do PSP ficavam armazenados em um banco de dados centralizado. Essa característica mostrava-se uma ótima oportunidade para que professores e pesquisadores analisassem estatisticamente os dados para descobrir tendências dos alunos, como por exemplo, verificar se o número médio de defeitos dos programas desenvolvidos diminuía conforme se desenvolviam atividades relacionadas à melhoria da qualidade pessoal. Assim, um módulo de análise de dados seria construído para que professores e pesquisadores, devidamente autenticados e autorizados, pudessem realizar tais análises em turmas específicas.

No entanto, quando se constatou que ferramentas de 2ª geração eram parte do problema de abandono do PSP, esse planejamento foi substituído por um novo planejamento envolvendo ferramentas de 3ª geração, o qual é apresentado na Seção 3.3. No entanto, decidiu-se apresentar esse planejamento inicial, pois durante alguns meses ele foi o foco deste trabalho e as atividades realizadas tinham esse outro objetivo. As Tabelas 4 a 9 retratam o planejamento inicial.

Tabela 4 – Planejamento Inicial (Objetivos em Alto Nível)

OBJETIVOS EM ALTO NÍVEL	
Pressuposto	Quanto melhores desenvolvedores os alunos se tornarem, maior será a qualidade de seu trabalho, tanto na academia quanto em empresas
Objetivos	**Tornar os alunos melhores desenvolvedores de software**
Atividade	Melhorar
Foco	As técnicas de desenvolvimento de software
Objeto	Alunos da Graduação em Ciência da Computação
Duração	Durante todo o curso de graduação
Contexto	Curso de Graduação em Ciência da Computação da Universidade Federal de São Carlos

Tabela 5 – Planejamento Inicial (Estratégia)

ESTRATÉGIA	
Pressuposto	Ao utilizarem o método PSP, a qualidade do trabalho dos alunos aumentará.
Decisões Estratégicas	**Implantar o PSP no curso de graduação.**
Contexto	Os alunos atualmente não utilizam nenhuma metodologia consistente de desenvolvimento.

Tabela 6 – Planejamento Inicial (Objetivos de Software)

OBJETIVOS DE SOFTWARE	
Pressuposto	As ferramentas de apoio ao PSP atualmente disponíveis não são totalmente apropriadas para o ensino do PSP em um curso de graduação.
Contexto	Formulários Manuais ou Ferramentas Computacionais que apóiam o método PSP.
Objetivo	**Elaborar uma nova ferramenta de apoio ao PSP.**
Atividade	Elaborar
Foco	Ferramenta PSP-TUTOR
Objeto	Personal Software Process
Duração	18 a 22 meses
Contexto	Curso de Mestrado em Ciência da Computação
Limitações	As tecnologias envolvidas na elaboração da ferramenta não são plenamente conhecidas pelo desenvolvedor; tempo máximo de duração é limitado por duração do curso de mestrado.

Tabela 7 – Planejamento Inicial (Objetivos de Medição)

OBJETIVOS DE MEDIÇÃO	
Pressuposto	A ferramenta PSP-TUTOR é adequada para o ensino do PSP no meio acadêmico e facilita a implantação do PSP no Curso de Graduação.
Contexto	Ferramenta PSP-TUTOR desenvolvida neste trabalho de mestrado.
Cenários Possíveis	Elaboração de uma ferramenta *"stand-alone"* (*Desktop*) em Java a) Desenvolver uma ferramenta utilizando Java Swing, JPA e Java WebStart que apóie todas as fases do PSP em formato "tutorial" b) Aplicar a ferramenta em disciplinas de graduação e avaliar a aceitação da ferramenta e a absorção dos conceitos do PSP por parte dos alunos.
	Elaboração de uma ferramenta "Web" (Cenário Escolhido) **a) Desenvolver uma ferramenta utilizando EJB, JPA, JSF e JAAS que apóie todas as etapas do PSP em formato "tutorial".** **b) Desenvolver um módulo de análise de dados para que professores e pesquisadores possam analisar o impacto do método PSP na qualidade e na produtividade dos alunos.** **c) Aplicar a ferramenta em disciplinas da graduação e avaliar a aceitação da ferramenta e a absorção dos conceitos do PSP por parte dos alunos.**

Tabela 8 – Planejamento Inicial (Objetivos de Métricas)

Objetivo G1	Objeto	Ferramentas de apoio ao PSP
	Propósito	Identificar
	Foco de Qualidade	Características positivas e negativas
	Ponto de Vista	Pesquisador
	Contexto	Elaboração de uma nova ferramenta de apoio ao PSP
Objetivo G2	Objeto	EJB, JSF, JAAS
	Propósito	Avaliar
	Foco de Qualidade	Adequação das tecnologias à construção da ferramenta
	Ponto de Vista	Desenvolvedor
	Contexto	Elaboração de uma nova ferramenta de apoio ao PSP
Objetivo G3	Objeto	Ferramenta PSP-TUTOR
	Propósito	Avaliar
	Foco de Qualidade	Adequação da ferramenta ao ensino do PSP
	Ponto de Vista	Pesquisador
	Contexto	Curso de Graduação em Ciência da Computação
Objetivo G4	Objeto	Alunos de Graduação em Ciência da Computação
	Propósito	Avaliar
	Foco de Qualidade	Qualidade do Processo de Desenvolvimento de Software
	Ponto de Vista	Pesquisador
	Contexto	Curso de Graduação em Ciência da Computação

Tabela 9 – Planejamento Inicial (Questões e Métricas definidas)

Questões	Métricas
Q1. Quais as características das ferramentas de apoio ao PSP atualmente disponíveis?	M1. Características descritivas M2. Custo M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M5. Benefícios encontrados M6. Dificuldades encontradas
Q2. Por que as ferramentas atualmente disponíveis não são apropriadas para o ensino do PSP em um curso de graduação?	M6. Dificuldades encontradas M7. Possíveis soluções para as dificuldades encontradas M8. Número de atividades do PSP com suporte M9. Número de atividades do PSP sem suporte
Q3. Quais as tecnologias mais apropriadas para o desenvolvimento de uma ferramenta *"Web"*?	M10. Tecnologias que têm se desenvolvido nos últimos dois anos
Q4. A ferramenta construída apóia, efetivamente, o PSP?	M8. Número de atividades do PSP com suporte M9. Número de atividades do PSP sem suporte
Q5. Os alunos, ao utilizarem a ferramenta construída, estão apresentando as características esperadas?	M11. Média de número de defeitos M12. LOC/hora M13. Erro no planejamento inicial de tempo M14. Erro no planejamento inicial de tamanho

A Figura 4 retrata o relacionamento entre os quatro objetivos (G), cinco perguntas (Q) e quatorze métricas (M):

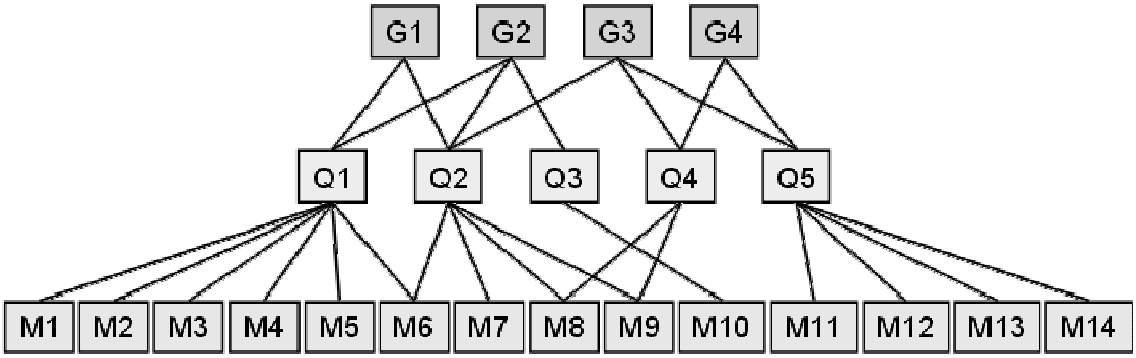


Figura 4 – Árvore GQM do Planejamento Inicial

Para determinar se os objetivos traçados no nível conceitual foram atingidos, foi elaborado o seguinte modelo de interpretação:

- Para cada ferramenta de apoio ao PSP atualmente disponível
  - Identificar M1, M2, ..., M9;
- Se M5 < M6 ou M8 < M9
  - Ferramenta não é apropriada para ensino de PSP para alunos de graduação;
- Se não houver ferramenta apropriada, construir uma nova ferramenta de apoio ou modificar uma que já está disponível
  - Identificar M10;
- Elaborar estudo de caso em que a ferramenta é aplicada
  - Se M11, M13 e M14 se mantêm constantes ou aumentam com o uso da ferramenta
    - Se M8 > M9
      - Reavaliar M7 e M10;
    - Se M5 < M6
      - Proposta da ferramenta falhou;
  - Se M12 diminui com o uso da ferramenta
    - Se M8 > M9
      - Reavaliar M7 e M10;
    - Se M5 < M6
      - Proposta da ferramenta falhou;

- Verificar se é possível identificar benefícios do uso da ferramenta e do PSP do ponto de vista dos professores das disciplinas
    - Em caso negativo, reavaliar todo planejamento GQM;

Seguindo esse planejamento, a partir do objetivo G1 temos que este se relaciona com as questões Q1 e Q2, que por sua vez se relacionam com as métricas M1 à M9. O objetivo G1 diz respeito a identificar os pontos positivos e negativos das ferramentas de apoio ao PSP então disponíveis, através de métricas como número de casos de sucesso e quantidade de atividades do PSP apoiadas. Assim, constatou-se que trabalhos da área identificaram que ferramentas de apoio ao PSP de 2ª geração apresentavam problemas de adoção em longo prazo, pois o desenvolvedor tinha que constantemente mudar entre os contextos de desenvolvedor de software e coletor de métricas. Essa constante troca de contexto causa muitas interrupções no fluxo de trabalho do desenvolvedor, sobrecarregando-o. Os problemas inerentes a ferramentas de 2ª geração são discutidos em detalhes na Seção 2.4.2.

Desta maneira, o desenvolvimento da ferramenta PSP-TUTOR foi cancelado, assim como todo o planejamento elaborado em torno do desenvolvimento dessa ferramenta. Foi determinante para essa decisão o fato de pesquisadores na Universidade do Havaí já terem previamente desenvolvido uma ferramenta de 2ª geração, aplicado-a em ambiente acadêmico e então registrado resultados não satisfatórios [Johnson et. al., 2003] passando, portanto, a desenvolver uma nova abordagem através do desenvolvimento de uma ferramenta de apoio de 3ª geração.

Assim, o foco deste trabalho passou a ser ferramentas de 3ª geração. Esta abordagem é relativamente recente e, portanto, pretende-se identificar as vantagens e as desvantagens de ferramentas de 3ª geração em relação a ferramentas de 1ª e 2ª gerações através de estudos de caso elaborados tanto em ambiente acadêmico quanto em empresas de software, planejados utilizando o paradigma GQM. A Seção 3.3 detalha o planejamento de tais estudos de caso.

3.3. Planejamento para a avaliação de ferramentas de 3ª. Geração

Ferramentas de 3ª geração caracterizam-se por serem capazes de coletar e analisar métricas de Engenharia de Software automaticamente. Essa característica faz

com que essas ferramentas ataquem diretamente o problema de abandono do PSP devido à sobrecarga das ferramentas de 2ª geração. No entanto, nem todas as atividades do PSP podem ser totalmente automatizadas, já que algumas atividades envolvem diretamente o julgamento humano. Assim, ferramentas de 3ª geração apresentam um novo cenário de apoio ao PSP, pois ao mesmo tempo que auxiliam no uso prolongado do PSP, nem todas as atividades do PSP são apoiadas.

Por apoiarem um limitado número de atividades do PSP, ferramentas de 3ª geração acabaram se distanciando da metodologia e passaram a ser classificadas como ferramentas ISEMA – *In-Process Software Engineering Measurement and Analysis* – ou seja, ferramentas que coletam e analisam automaticamente métricas de Engenharia de Software de projetos em andamento. Essa abordagem difere da abordagem do PSP, pois neste são utilizados dados de projetos anteriores para planejar projetos futuros, enquanto ferramentas ISEMA utilizam dados do próprio projeto em andamento para o planejamento futuro. Ferramentas de 3ª geração são apresentadas em maiores detalhes na Seção 2.4.3.

Dentro do contexto exposto, foram elaborados neste trabalho dois estudos de caso com o auxílio do paradigma GQM que têm o objetivo de identificar os pontos positivos e os pontos negativos da abordagem proposta por ferramentas de 3ª geração. Um estudo de caso foi elaborado em ambiente acadêmico com alunos de Graduação em Ciência da Computação e o seu planejamento é apresentado na Seção 3.3.1. O outro estudo de caso foi elaborado em uma pequena empresa de desenvolvimento de software e seu planejamento é apresentado na Seção 3.3.2.

3.3.1. Estudo de Caso em Ambiente Acadêmico

No planejamento inicial discutido na Seção 3.2, o principal objetivo era a implantação do PSP em todo o curso de graduação em Ciência da Computação da Universidade Federal de São Carlos. Mesmo que o planejamento tenha mudado o foco de ferramentas de 2ª geração para ferramentas de 3ª geração, este objetivo se mantém inalterado. De fato, muitos dos objetivos, questões e métricas do planejamento inicial também se apresentam neste planejamento. Assim, o estudo de caso planejado segundo o paradigma GQM para o Ambiente Acadêmico tenta identificar os benefícios e as dificuldades do uso de ferramentas de 3ª geração no apoio ao uso de PSP por alunos de

graduação A tabelas 10 à 15 retratam o planejamento elaborado para o ambiente acadêmico.

Tabela 10 – Planejamento em Ambiente Acadêmico (Objetivos em Alto Nível)

OBJETIVOS EM ALTO NÍVEL	
Pressuposto	Quanto melhores desenvolvedores os alunos se tornarem, maior será a qualidade de seu trabalho, tanto na academia quanto em empresas
Objetivos	**Tornar os alunos melhores desenvolvedores de software**
Atividade	Melhorar
Foco	As técnicas de elaboração de software
Objeto	Alunos da Graduação em Ciência da Computação
Duração	Durante todo o curso de graduação
Contexto	Curso de Graduação em Ciência da Computação da Universidade Federal de São Carlos

Tabela 11 – Planejamento em Ambiente Acadêmico (Estratégia)

ESTRATÉGIA	
Pressuposto	Ao utilizarem o método PSP, a qualidade do trabalho dos alunos aumentará.
Decisões Estratégicas	**Implantar o PSP no curso de graduação.**
Contexto	Os alunos atualmente não utilizam nenhuma metodologia consistente de desenvolvimento.

Tabela 12 – Planejamento em Ambiente Acadêmico (Objetivos de Software)

OBJETIVOS DE SOFTWARE	
Pressuposto	Ferramentas de 3ª geração são apropriadas para apoiar a implantação do PSP em um curso de graduação em Ciência da Computação.
Contexto	Ferramentas de 3ª geração coletam e analisam métricas de Engenharia de Software automaticamente, sem intervenção do desenvolvedor.
Objetivo	**Utilizar ferramentas de 3ª geração para apoiar a aplicação do PSP pelos alunos**
Atividade	Apoiar
Foco	Ferramentas de apoio ao PSP de 3ª geração
Objeto	Personal Software Process
Duração	Durante todo o curso
Contexto	Curso de Graduação em Ciência da Computação
Limitações	Nem todas as disciplinas de programação podem utilizar ambientes de programação compatíveis com ferramentas de 3ª geração.

Tabela 13 – Planejamento em Ambiente Acadêmico (Objetivos de Medição)

OBJETIVOS DE MEDIÇÃO	
Pressuposto	O uso do PSP apoiado por ferramentas de 3ª geração em ambiente acadêmico não apresentará os mesmos problemas de abandono em longo prazo verificados em ferramentas de 2ª geração.
Contexto	Alunos do curso de graduação que nunca utilizaram o PSP
Cenários Possíveis	Aplicar o PSP em uma turma especialmente voltada ao PSP, em que a participação no estudo de caso influencia diretamente na avaliação do aluno na disciplina.
	(Cenário escolhido) **Aplicar o PSP em uma turma regular em que são exercidas atividades de desenvolvimento de software. A participação no estudo de caso é voluntária e não está vinculada à nota final da disciplina.**

Tabela 14 – Planejamento em Ambiente Acadêmico (Objetivos de Métricas)

Objetivo G1	Objeto	Ferramenta *Hackystat*
	Propósito	Identificar
	Foco de Qualidade	Características positivas e negativas
	Ponto de Vista	Pesquisador
	Contexto	Dissertação de Mestrado
Objetivo G2	Objeto	PSP
	Propósito	Avaliar
	Foco de Qualidade	Número de atividades do PSP apoiadas pela *Hackystat*
	Ponto de Vista	Pesquisador
	Contexto	Dissertação de Mestrado
Objetivo G3	Objeto	Ferramenta *Hackystat*
	Propósito	Avaliar
	Foco de Qualidade	Adequação da ferramenta em Ambiente Acadêmico
	Ponto de Vista	Pesquisador
	Contexto	Curso de Graduação em Ciência da Computação
Objetivo G4	Objeto	Alunos de Graduação em Ciência da Computação
	Propósito	Avaliar
	Foco de Qualidade	Qualidade do Processo de Desenvolvimento de Software
	Ponto de Vista	Pesquisador
	Contexto	Curso de Graduação em Ciência da Computação

Tabela 15 – Planejamento em Ambiente Acadêmico (Questões e Métricas)

Questões	Métricas
Q1. Quais as características das ferramentas de 3ª geração que a diferenciam de ferramentas de gerações anteriores?	M1. Características descritivas M2. Custo M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M5. Benefícios encontrados M6. Dificuldades encontradas M7. Número de atividades do PSP com suporte M8. Número de atividades do PSP sem suporte
Q2. Quais são as atividades do PSP com suporte?	M7. Número de atividades do PSP com suporte M8. Número de atividades do PSP sem suporte
Q3. Quais os ambientes de programação mais utilizados pelos alunos?	M9. Lista de ambientes de programação mais utilizados pelos alunos
Q4. Qual a precisão do planejamento de tempo dos alunos?	M10. Percentual de alunos que realizam estimativas de tempo. M11. Percentual de alunos que têm algum sucesso em estimativas de tempo M12. Métodos de estimativas de tempo mais utilizadas pelos alunos
Q5. Quais atividades os alunos realizam com o intuito de aumentar a qualidade de seu trabalho?	M13. Número de alunos que se preocupam com a qualidade de seu trabalho. M14. Metodologias utilizadas pelos alunos para aumentar a qualidade de seu trabalho M15. Porcentagem de melhoria registrada pelos alunos
Q6. Qual o nível de interesse dos alunos em metodologias de desenvolvimento de software e atividades de Engenharia de Software?	M16. Porcentagem de alunos que têm interesse em aprender atividades de Engenharia de Software. M17. Porcentagem de alunos que julgam atividades de Engenharia de Software importantes para seu desenvolvimento acadêmico e profissional.
Q7. Qual o conhecimento dos alunos sobre a metodologia PSP?	M18. Número de alunos que já estudaram o PSP. M19. Número de alunos que já utilizaram o PSP. M20. Número de alunos que já participaram de um treinamento formal sobre o PSP. M21. Níveis do PSP em que os alunos se encaixam
Q8. Qual o interesse dos alunos na metodologia PSP?	M22. Percentual de alunos que voltariam a utilizar o PSP espontaneamente. M23. Percentual de alunos que dizem ter tido uma experiência positiva com o PSP. M24. Lista dos principais motivos para os alunos não usarem o PSP
Q9. Qual é a interferência da ferramenta de 3ª geração na aplicação do PSP?	M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M7. Número de atividades do PSP com suporte M8. Número de atividades do PSP sem suporte M25. Percentual de alunos que julgam que a ferramenta auxiliou na aplicação do PSP. M26. Percentual de alunos que julgam que a ferramenta atrapalhou a aplicação do PSP.
Q10. Qual é a relação da abordagem do uso do PSP com ferramentas de 3ª geração na qualidade do trabalho dos alunos?	M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M5. Benefícios encontrados M6. Dificuldades encontradas M27. Percentual de alunos que não registraram qualquer impacto. M28. Percentual de alunos que registraram melhorias nas estimativas. M29. Percentual de alunos que registraram diminuição do número médio de defeitos de seus programas M30. Percentual de alunos que registraram que a sua produtividade aumentou com o uso do PSP M31. Percentual de alunos que registraram um impacto negativo em seu trabalho com o uso do PSP.

A Figura 5 retrata o relacionamento entre os quatro objetivos (G), dez perguntas (Q) e trinta e uma métricas (M) presentes nesse planejamento:

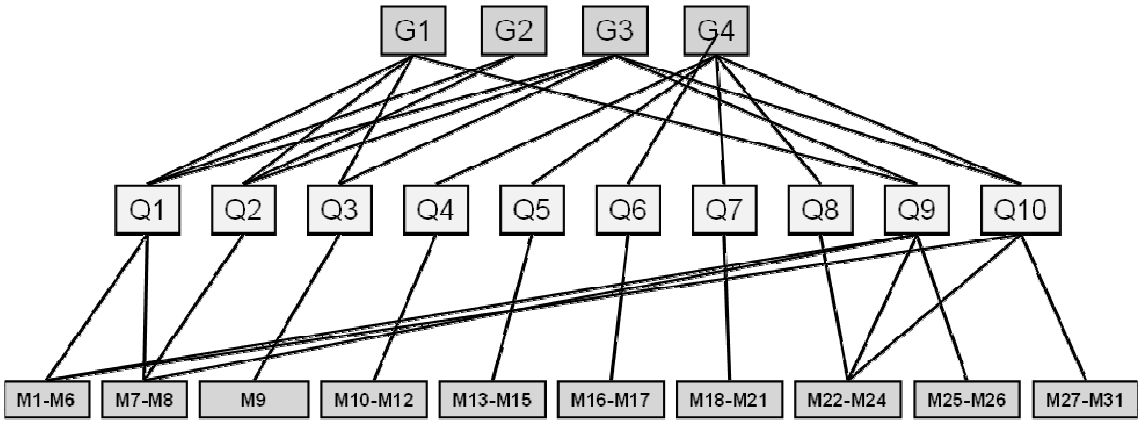


Figura 5 – Árvore GQM do Planejamento em Ambiente Acadêmico

Para determinar se os objetivos traçados no nível conceitual foram atingidos, foi elaborado o seguinte modelo de interpretação:

- Para cada ferramenta de 3ª geração disponível, identificar M1 à M9;
- Aplicar a ferramenta de 3ª geração mais apropriada nas turmas de graduação em Ciência da Computação
    - Se o estudo de caso registrar valores de M4, M15 e M23 inferiores aos valores encontrados em trabalhos relacionados
        - Verificar se ferramenta é apropriada para ambiente acadêmico;
    - Se os valores iniciais de M10, M11, M13, M15, M16, M17, M22, M23, M28, M29 e M30 forem superiores aos valores registrados após o estudo de caso
        - Benefícios esperados do PSP não foram registrados. Rever abordagem;

Durante o planejamento, já se sabia que o tempo disponível para a realização dos estudos de caso era bastante limitado. Tal limitação existe, pois se deve obedecer aos prazos máximos do Programa de Pós-Graduação em Ciência da Computação. Essa limitação de tempo dificulta ainda mais a interpretação de resultados, pois como os estudos de caso são realizados em turmas regulares do curso de graduação em Ciência da Computação, as limitações de tempo implicam em limitação do número de dados que podem ser coletados. Com mais tempo disponível, mais turmas poderiam participar do

estudo de caso e, assim, mais hipóteses poderiam ser avaliadas. A indisponibilidade de um maior período de tempo para a execução dos estudos de caso é, portanto, de grande impacto no planejamento. No entanto, optou-se por elaborar o planejamento sem levar em consideração tal fato, pois se pretende que trabalhos futuros complementem os estudos de caso apresentados nos Capítulos 4 e 5. Assim, acredita-se que o planejamento, como apresentado nessa Seção, é de valiosa utilidade para a realização de trabalhos futuros na área.

3.3.2. Estudo de Caso na Pequena Empresa

O planejamento do estudo de caso para a pequena empresa inclui os objetivos citados no planejamento do estudo de caso em ambiente acadêmico (Seção 3.3.1), mas adaptados ao cenário de uma pequena empresa de desenvolvimento de software. Ou seja, um dos objetivos é identificar os benefícios e dificuldades do uso de ferramentas de apoio ao PSP de 3ª geração na indústria de software.

A empresa em que o estudo de caso se realizaria já utilizava o PSP em seus projetos de desenvolvimento de software, sendo que seus desenvolvedores realizam atividades do Nível 1.1. Para apoiar a aplicação do PSP, era utilizada a ferramenta de 2ª geração *Process Dashboard*. Como ferramentas de 3ª geração coletam métricas de Engenharia de Software automaticamente e sem a intervenção do desenvolvedor, espera-se que a qualidade das métricas coletadas seja maior do que a qualidade de métricas coletadas por ferramentas de 2ª geração [Johnson & Disney, 1998; Johnson, 2003]. Assim, mais um objetivo foi adicionado ao planejamento desse estudo de caso, com o intuito de verificar se há diferenças relevantes entre as métricas coletadas por ferramentas de diferentes gerações.

Para avaliar esse objetivo, os desenvolvedores continuariam a utilizar o PSP com ferramentas de 2ª geração, mas também usariam ferramentas de 3ª geração simultaneamente. Desta maneira era possível coletar as mesmas métricas das duas maneiras, nos mesmos projetos e nas mesmas condições. Tal abordagem não sobrecarrega os desenvolvedores, pois ferramentas de 3ª geração são capazes de coletar e analisar métricas de Engenharia de Software de modo não obstrutivo aos desenvolvedores. A formalização do planejamento, segundo o paradigma GQM, é apresentada nas tabelas 16 a 21.

Tabela 16 – Planejamento em Empresas (Objetivos em Alto Nível)

OBJETIVOS EM ALTO NÍVEL	
Pressuposto	A qualidade do trabalho pessoal dos desenvolvedores de software reflete na qualidade do trabalho da empresa como um todo
Objetivos	**Tornar os profissionais da indústria melhores desenvolvedores de software**
Atividade	Melhorar
Foco	As técnicas de desenvolvimento de software
Objeto	Profissionais da indústria de software
Duração	Contínua
Contexto	Pequena empresa de desenvolvimento de software

Tabela 17 – Planejamento em Empresas (Estratégia)

ESTRATÉGIA	
Pressuposto	Ao utilizarem o método PSP, a qualidade do trabalho dos desenvolvedores aumentará.
Decisões Estratégicas	**Implantar o PSP na empresa de desenvolvimento de software.**
Contexto	Pequena empresa de desenvolvimento de software na cidade de São Carlos.

Tabela 18 – Planejamento em Empresas (Objetivos de Software)

OBJETIVOS DE SOFTWARE	
Pressuposto	Ferramentas de 3ª geração são apropriadas para apoiar a aplicação do PSP em empresas de desenvolvimento de software
Contexto	A empresa atualmente utiliza o PSP apoiado por ferramentas de 2ª geração
Objetivo	**Utilizar ferramentas de 3ª geração para apoiar a aplicação do PSP pelos desenvolvedores**
Atividade	Apoiar
Foco	Ferramentas de apoio ao PSP de 3ª geração
Objeto	Personal Software Process
Duração	Contínua
Contexto	Pequena Empresa de Desenvolvimento de Software
Limitações	A coleta dos dados é limitada por tempo máximo de curso de mestrado.

Tabela 19 – Planejamento em Empresas (Objetivos de Medição)

OBJETIVOS DE MEDIÇÃO	
Pressuposto	O uso do PSP apoiado por ferramentas de 3ª geração na indústria de desenvolvimento de software não apresentará a mesma resistência por parte dos desenvolvedores verificada com ferramentas de 2ª geração.
Contexto	Pequena empresa de desenvolvimento de software
Cenários Possíveis	Substituir ferramentas de 2ª geração por ferramentas de 3ª geração.
	(Cenário escolhido) **Utilizar ferramentas de 2ª e de 3ª gerações simultaneamente.**

Tabela 20 – Planejamento em Empresas (Objetivos de Métricas)

Objetivo G1	Objeto	Ferramenta *Hackystat*
	Propósito	Identificar
	Foco de Qualidade	Características positivas e negativas
	Ponto de Vista	Pesquisador
	Contexto	Dissertação de Mestrado
Objetivo G2	Objeto	PSP
	Propósito	Avaliar
	Foco de Qualidade	Número de atividades do PSP apoiadas pela *Hackystat*
	Ponto de Vista	Pesquisador
	Contexto	Dissertação de Mestrado
Objetivo G3	Objeto	Ferramenta *Hackystat*
	Propósito	Avaliar
	Foco de Qualidade	Adequação da ferramenta ao Ambiente Industrial
	Ponto de Vista	Pesquisador
	Contexto	Pequena Empresa de Desenvolvimento de Software
Objetivo G4	Objeto	Ferramenta *Hackystat*
	Propósito	Comparar
	Foco de Qualidade	Coleta de métricas em relação a ferramentas de 2ª geração
	Ponto de Vista	Pesquisador
	Contexto	Pequena Empresa de Desenvolvimento de Software
Objetivo G5	Objeto	Desenvolvedores de Software
	Propósito	Avaliar
	Foco de Qualidade	Qualidade do Processo de Desenvolvimento de Software
	Ponto de Vista	Pesquisador
	Contexto	Pequena empresa de desenvolvimento de software

Tabela 21 - Planejamento em Empresas (Questões e Métricas)

Questões	Métricas
Q1. Quais as características das ferramentas de 3ª geração que a diferenciam de ferramentas de gerações anteriores?	M1. Características descritivas M2. Custo M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M5. Benefícios encontrados M6. Dificuldades encontradas M7. Número de atividades do PSP com suporte M8. Número de atividades do PSP sem suporte
Q2. Qual o número de atividades do PSP com suporte?	M7. Número de atividades do PSP com suporte M8. Número de atividades do PSP sem suporte
Q3. Quais os ambientes de programação mais utilizados pelos desenvolvedores?	M9. Lista de ambientes de programação mais utilizados pelos desenvolvedores
Q4. Qual a taxa de acerto de estimativas de tempo dos desenvolvedores?	M10. Percentual de desenvolvedores que realizam estimativas de tempo. M11. Percentual de desenvolvedores que têm algum sucesso em estimativas de tempo M12. Métodos de estimativas de tempo mais utilizadas pelos desenvolvedores

Q5. Quais atividades os desenvolvedores realizam com o intuito de aumentar a qualidade de seu trabalho?	M13. Número de desenvolvedores que se preocupam com a qualidade de seu trabalho. M14. Metodologias utilizadas pelos desenvolvedores para aumentar a qualidade de seu trabalho M15. Porcentagem de melhoria registrada pelos desenvolvedores
Q6. Qual o nível de interesse dos desenvolvedores em metodologias de desenvolvimento de software e atividades de Engenharia de Software?	M16. Porcentagem de desenvolvedores que têm interesse em aprender atividades de Engenharia de Software. M17. Porcentagem de desenvolvedores que julgam atividades de Engenharia de Software importantes para seu desenvolvimento profissional.
Q7. Qual o conhecimento dos desenvolvedores sobre a metodologia PSP?	M18. Número de desenvolvedores que já estudaram o PSP. M19. Número de desenvolvedores que já utilizaram o PSP. M20. Número de desenvolvedores que já participaram de um treinamento formal sobre o PSP. M21. Níveis do PSP em que os desenvolvedores se encaixam
Q8. Qual o interesse dos desenvolvedores na metodologia PSP?	M22. Percentual de desenvolvedores que voltariam a utilizar o PSP espontaneamente. M23. Percentual de desenvolvedores que dizem ter tido uma experiência positiva com o PSP. M24. Lista dos principais motivos citados pelos desenvolvedores para não usarem o PSP
Q9. Qual o impacto da ferramenta de 3ª geração na aplicação do PSP?	M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M7. Número de atividades do PSP com suporte M8. Número de atividades do PSP sem suporte M25. Percentual de desenvolvedores que julgam que a ferramenta auxiliou na aplicação do PSP M26. Percentual de desenvolvedores que julgam que a ferramenta atrapalhou a aplicação do PSP
Q10. Qual o impacto da abordagem do uso do PSP com ferramentas de 3ª geração na qualidade do trabalho dos alunos?	M3. Casos de Sucesso M4. Porcentagem de aceitação dos usuários M5. Benefícios encontrados M6. Dificuldades encontradas M27. Percentual de desenvolvedores que não registraram qualquer impacto. M28. Percentual de desenvolvedores que registraram melhorias nas estimativas de tempo. M29. Percentual de desenvolvedores que registraram diminuição do número médio de defeitos de seus programas M30. Percentual de desenvolvedores que registraram que a sua produtividade aumentou com o uso do PSP M31. Percentual de desenvolvedores que registraram um impacto negativo em seu trabalho com o uso do PSP.
Q11. As métricas coletadas por ferramentas de 3ª geração são mais precisas do que métricas coletadas por ferramentas de gerações anteriores?	M32. Lista de métricas que podem ser coletadas por todos os tipos de ferramentas. M33. Lista de métricas que não podem ser coletadas por ferramentas de 3ª geração. M34. Diferença registrada entre as métricas coletadas por diferentes gerações de ferramentas.

A Figura 6 retrata o relacionamento entre os objetivos (G), questões (Q) e métricas (M). Como o planejamento para ambiente empresarial engloba o planejamento

para ambiente acadêmico adaptado para a realidade de uma pequena empresa de desenvolvimento de software, a árvore GQM para o planejamento empresarial é idêntica à árvore GQM do planejamento para ambiente acadêmico, com exceção da inclusão do Objetivo G5, da Questão Q11 e das Métricas M32, M33 e M34. A Figura 6 resume a árvore GQM do planejamento para ambiente acadêmico, com os novos elementos inseridos (G5, Q11, M32-34).

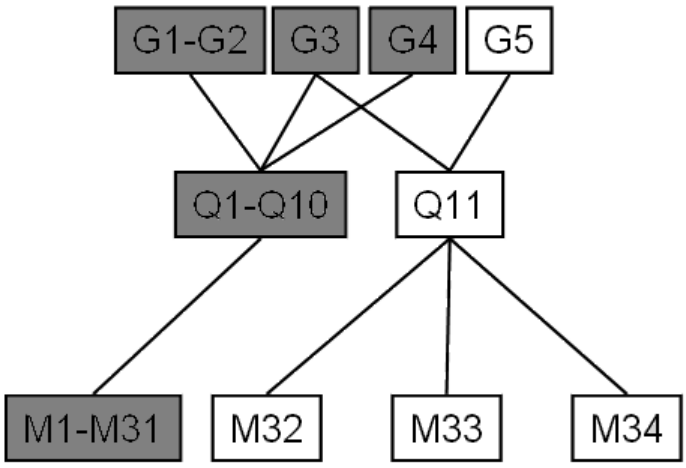


Figura 6 – Árvore GQM para o Planejamento em Ambiente Empresarial

O modelo de interpretação dos dados elaborado para o ambiente empresarial é semelhante ao modelo elaborado para o ambiente acadêmico:

- Para cada ferramenta de 3ª geração disponível, identificar M1 à M9 e M32 à M34;
- Aplicar a ferramenta de 3ª geração mais apropriada na empresa de desenvolvimento de software, mantendo o uso da ferramenta de 2ª geração
    - Se o estudo de caso registrar valores de M4, M15 e M23 inferiores aos valores encontrados em trabalhos relacionados
        - Verificar se ferramenta é apropriada para ambiente empresarial;
    - Se os valores iniciais de M10, M11, M13, M15, M16, M17, M22, M23, M28, M29 e M30 forem superiores aos valores registrados após o estudo de caso
        - Benefícios esperados do PSP não foram registrados. Rever abordagem;

- Comparar os valores de M10, M11, M13, M15, M16, M17, M22, M23, M28, M29 e M30 registrados em ambas as ferramentas. Se os valores se equivalem
  - Ferramentas de 3ª geração não têm sucesso em coletar métricas de maneira mais precisa do que ferramentas de gerações anteriores;

Esse modelo consiste em uma adaptação do modelo elaborado para ambiente acadêmico, acrescentado da interpretação dos dados em relação à coleta de métricas por ferramentas de diferentes gerações. Esse modelo tem as mesmas limitações discutidas na Seção 3.3.1 sobre o modelo elaborado para ambiente acadêmico. É essencial que para a aplicação integral desse modelo os desenvolvedores utilizem ferramentas de 2ª e 3ª gerações simultaneamente.

3.4. Considerações Finais

Neste capítulo foram apresentados os planejamentos dos estudos de caso presentes neste trabalho. Estudos de caso são importantes na investigação de entidades ou fenômenos, pois permitem, através de observação, que se compreendam as relações entre diversos dados. Mas como em um estudo de caso existem muitas variáveis, o paradigma GQM foi utilizado para auxiliar durante a fase de planejamento, especialmente na identificação das métricas que necessitam ser coletadas. Assim, o uso do GQM poupou que recursos fossem desperdiçados coletando métricas desnecessárias, ao mesmo tempo em que evitou que métricas importantes fossem ignoradas, o que dificultaria a análise dos dados.

Mesmo que as métricas corretas sejam coletadas, ainda assim não é trivial a análise dos dados, principalmente por dois motivos: falta de informações históricas, uma vez que esta é a primeira vez que um estudo de caso com essas características é elaborado no Departamento de Computação, e dificuldade em realizar comparações com outros estudos de caso de trabalhos da área, pois os estudos de caso apresentam características diferentes. Assim, optou-se por analisar os dados comparando o valor inicial das métricas (coletadas antes do início do estudo de caso) com o valor coletado ao final do estudo de caso, tentando, assim, detectar se a abordagem proposta por ferramentas de 3ª geração causam algum impacto, seja positivo ou negativo, no processo de desenvolvimento de software.

Na Seção 3.2 foi apresentado o planejamento inicial deste trabalho. Assim que se constatou que ferramentas de 2ª geração apresentavam problemas de adoção em longo prazo, o desenvolvimento da ferramenta de 2ª geração PSP-TUTOR foi cancelado, e o planejamento precisou ser revisto. Finalmente, na Seção 3.3 foram apresentados os planejamentos que foram elaborados para os dois estudos de caso que foram realizados, respectivamente, em ambiente acadêmico e em uma pequena empresa de desenvolvimento de software. Embora os planejamentos sejam similares, o estudo de caso em ambiente empresarial tem como objetivo adicional determinar se as métricas coletadas por ferramentas de 3ª geração são mais precisas do que métricas coletadas por ferramentas de 3ª geração. Tal comparação só foi possível pois os desenvolvedores da empresa utilizariam simultaneamente as duas ferramentas. Os Capítulos 4 e 5 apresentam os estudos de caso em maiores detalhes.

CAPÍTULO 4

ESTUDO DE CASO 1: AMBIENTE ACADÊMICO

4.1. Considerações Iniciais

O Estudo de Caso 1, apresentado neste capítulo, foi executado em Ambiente Acadêmico, segundo o planejamento discutido na Seção 3.3.1, e aplicado simultaneamente em duas turmas de disciplinas de desenvolvimento de software da Universidade Federal de São Carlos.

Dado que o principal objetivo do Estudo de Caso foi identificar os benefícios e as dificuldades do uso de ferramentas de apoio ao PSP de 3ª geração em ambiente acadêmico, é possível assim estabelecer um planejamento para que o PSP seja implantado em todas as disciplinas de desenvolvimento de software dos cursos de graduação em Ciência e Engenharia da Computação. Para isso, optou-se por elaborar o Estudo de Caso em disciplinas de desenvolvimento de software regularmente oferecidas pelo Departamento de Computação da Universidade Federal de São Carlos, ao invés de preparar uma turma específica para a aplicação do PSP.

O contexto do Estudo de Caso 1, incluindo a caracterização da população e as ferramentas utilizadas, é apresentado em detalhes na Seção 4.2. Os detalhes da execução desse estudo são apresentados na Seção 4.3 e os resultados são discutidos na Seção 4.4. As dificuldades identificadas durante a execução do Estudo de Caso 1 são discutidas na Seção 4.5, juntamente com os benefícios encontrados. Por fim, na Seção 4.6 são apresentadas as considerações finais do Capítulo.

4.2. Contexto do Estudo de Caso

O Estudo de Caso apresentado neste Capítulo foi executado simultaneamente em duas turmas de disciplinas com várias atividades de desenvolvimento de software da Universidade Federal de São Carlos. Embora as duas turmas tenham muitas características em comum, como as ferramentas utilizadas, a linguagem de programação

e o mesmo professor, algumas características particulares a cada turma são importantes de serem apresentadas para a correta análise dos resultados.

A Turma A era composta por alunos menos experientes com programação. Tais alunos desenvolveram um número menor de exercícios em que o PSP e a ferramenta *Hackystat* puderam ser utilizados, e contaram com relativamente pouco tempo para a instalação e configuração dessa ferramenta. As características da Turma A são mostrados em detalhes na Seção 4.2.1.

A Turma B era composta por alunos com mais experiência de programação do que a Turma A. Essa turma também realizou mais atividades em que o PSP e a ferramenta *Hackystat* foram utilizados, e ainda contou com mais tempo para instalar a IDE Eclipse e a ferramenta *Hackystat*

Ambas as turmas utilizaram a linguagem C++, o Ambiente de Desenvolvimento Integrado (IDE) Eclipse, a ferramenta de 3ª geração *Hackystat*, e eram orientadas pelo mesmo professor. Outras turmas de programação foram convidadas a participar do Estudo de Caso, mas devido a incompatibilidades de seus ambientes de desenvolvimento com a ferramenta *Hackystat* isso não foi possível.

Antes de iniciar o Estudo de Caso, uma palestra sobre o PSP e a ferramenta *Hackystat* foi apresentada aos alunos para que ficassem claros todos os conceitos e atividades que seriam realizadas no Estudo de Caso. Também foi apresentado um exemplo de uso do PSP com a ferramenta *Hackystat* exatamente como os alunos fariam em seu primeiro programa. Uma disciplina foi criada no ambiente de aprendizado Moodle [MOODLE, 2008] para apoiar o Estudo de Caso, no qual se disponibilizaram vários documentos e tutoriais, e por meio do qual dúvidas podiam ser tiradas pelos alunos.

Nessa palestra foi reforçado aos alunos que a participação era voluntária, que as atividades desenvolvidas durante o Estudo de Caso não interfeririam, em absolutamente nenhuma hipótese, na nota final da disciplina, e que os dados fornecidos pelos alunos da turma seriam analisados e publicados em Dissertação de Mestrado. Após a palestra, ambas as turmas responderam a um questionário anônimo em que as respostas foram utilizadas para a caracterização da população que participaria do Estudo de Caso. O questionário foi elaborado de forma a impedir a identificação dos alunos, para que os

dados coletados não sofressem distorções provocadas por receio dos alunos serem identificados. O questionário para caracterização da população é apresentado no Apêndice A.

A ferramenta *Hackystat*, utilizada no Estudo de Caso, é um software Cliente-Servidor e, portanto, é preciso que os alunos instalem um software em seus computadores e o configurem para utilizar os recursos disponibilizados em um servidor. Havia duas opções quanto à disponibilidade do servidor: instalar e configurar um servidor *Hackystat* no Departamento de Computação, ou então utilizar o servidor público e gratuito disponibilizado pela Universidade do Havaí. Escolheu-se a segunda opção, pois a instalação e configuração de um servidor *Hackystat* não são tarefas triviais, e problemas com o servidor colocariam em risco todo o Estudo de Caso. Tal escolha, no entanto, tem alguns pontos negativos, discutidos na Seção 4.5

4.2.1. Caracterização da Turma A

A Turma A era composta por alunos da disciplina "Programação de Computadores", cujo principal objetivo é ensinar conceitos de Programação Orientada a Objetos (POO) e a linguagem de programação C++. No total, 21 alunos estavam inscritos na turma, sendo que 20 alunos responderam ao questionário de caracterização da população antes do início do Estudo de Caso.

Tabela 22 – Experiência dos alunos como desenvolvedores de software (Turma A)

Experiência em Ambiente de Empresa	Percentagem de Alunos	Experiência em Ambiente Acadêmico	Percentagem de Alunos
Sem Experiência	95%	**Sem Experiência (0 a 7 programas)**	20%
Pouca Experiência (até 1 ano)	-	**Pouca Experiência (8 a 15 programas)**	45%
Experiência Intermediária (1 a 2 anos)	-	**Experiência intermediária (16 a 20 programas)**	30%
Muita Experiência (2 anos ou mais)	5%	**Muita Experiência (21 ou mais programas)**	5%

Uma das importantes características identificadas foi a experiência dos alunos como programadores de software. Procurou-se, então, determinar quantos programas os alunos desenvolveram em sua vida acadêmica e quantos anos de experiência na

indústria de desenvolvimento de software eles possuíam. A Tabela 22 resume os dados coletados.

O perfil da Turma A mostra que os alunos, de maneira geral, não possuíam muita experiência como desenvolvedores de software. Esse perfil já era esperado, pois a disciplina é ministrada no segundo semestre do curso de Ciência da Computação. Alunos que relataram maior experiência podem ter sido transferidos de outros cursos ou Instituições Superiores de Ensino, e assim apresentam um número maior de programas desenvolvidos. Todos os alunos estavam cursando a disciplina pela primeira vez, pois relataram que estavam cursando o primeiro ano da graduação. A turma estava dividida em 52% alunos de Ciência da Computação e 48% de alunos da Engenharia da Computação.

Os alunos dessa turma, por determinação do professor, utilizariam a IDE Eclipse para desenvolverem os projetos de software relacionados à disciplina, pois este era requisito da disciplina. Esse requisito foi bastante positivo para a realização do Estudo de Caso, pois a IDE Eclipse é compatível com a ferramenta *Hackystat*. Mesmo sabendo que os alunos utilizariam a IDE Eclipse, procurou-se identificar quais as IDEs que eles estavam acostumados a usar. A Figura 7 mostra que nenhum aluno costumava utilizar a IDE Eclipse até o momento, sendo que as IDEs mais utilizadas eram a DevC++/DevPascal (54,3%), Microsoft Visual Studio (14,3%) e Vi/Vim (11,4%). Outras IDEs citadas incluem Delphi, Turbo Pascal, Dreamweaver, FrontPage, Zend e GNU Fortran.

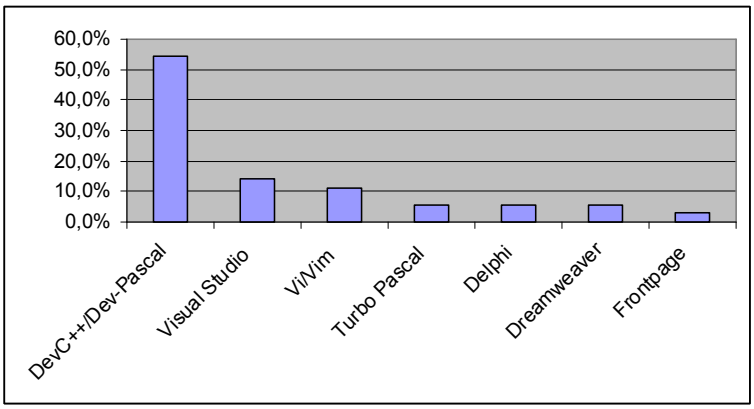


Figura 7 – IDEs habitualmente utilizadas pelos alunos (Turma A)

A Turma A desenvolveu três programas que fizeram parte do Estudo de Caso. Essa turma teve disponível duas semanas para instalar e configurar a IDE Eclipse e a ferramenta *Hackystat* antes que a primeira atividade relacionada ao Estudo de Caso se iniciasse.

4.2.2. Caracterização da Turma B

A Turma B era composta por alunos da disciplina Estrutura de Dados, que utilizava conceitos de POO e a linguagem C++. No total, 41 alunos estavam inscritos na turma, sendo que 38 responderam ao questionário de caracterização da população antes do início do Estudo de Caso. O questionário de caracterização da Turma B, identificando o nível de experiência dos alunos como desenvolvedores de software, é apresentado na Tabela 23.

Tabela 23 – Experiência dos alunos como desenvolvedores de software (Turma B)

Experiência em Ambiente de Empresa	Percentagem de Alunos	Experiência em Ambiente Acadêmico	Percentagem de Alunos
Sem Experiência	94,7%	**Sem Experiência (0 a 7 programas)**	-
Pouca Experiência (até 1 ano)	2,6%	**Pouca Experiência (8 a 15 programas)**	34,2%
Experiência Intermediária (1 a 2 anos)	2,6%	**Experiência intermediária (16 a 20 programas)**	44,7%
Muita Experiência (2 anos ou mais)	-	**Muita Experiência (21 ou mais programas)**	21,0%

O perfil da Turma B mostra que, embora os alunos apresentassem praticamente a mesma experiência que os alunos da Turma A em Ambiente Industrial, eles possuíam mais experiência como desenvolvedores de software em Ambiente Acadêmico. Esse perfil também já era esperado, pois a disciplina é ministrada no terceiro semestre do curso de Ciência da Computação e essa turma, em especial, foi criada para os alunos que já haviam cursado a disciplina anteriormente, mas não haviam conseguido aprovação. Portanto, é natural que tais alunos tivessem desenvolvido um número maior de programas do que alunos da Turma A. A Turma B era dividida em 60% de alunos de Ciência da Computação e 40% de alunos da Engenharia da Computação.

Os alunos dessa turma também tinham como requisito da disciplina o uso da IDE Eclipse para desenvolverem os seus projetos de software. Analogamente à Turma

A, procurou-se identificar quais as IDEs que os alunos estavam acostumados a usar. A Figura 8 mostra que 12,2% dos alunos já usavam o Eclipse como Ambiente de Desenvolvimento, o que tornou a sua aceitação pela turma consideravelmente mais fácil do que na Turma A. No entanto, mais uma vez, as IDEs DevC++/DevPascal (46,3%) e Microsoft Visual Studio (13,4%) foram as IDEs que os alunos estavam mais habituados a usar.

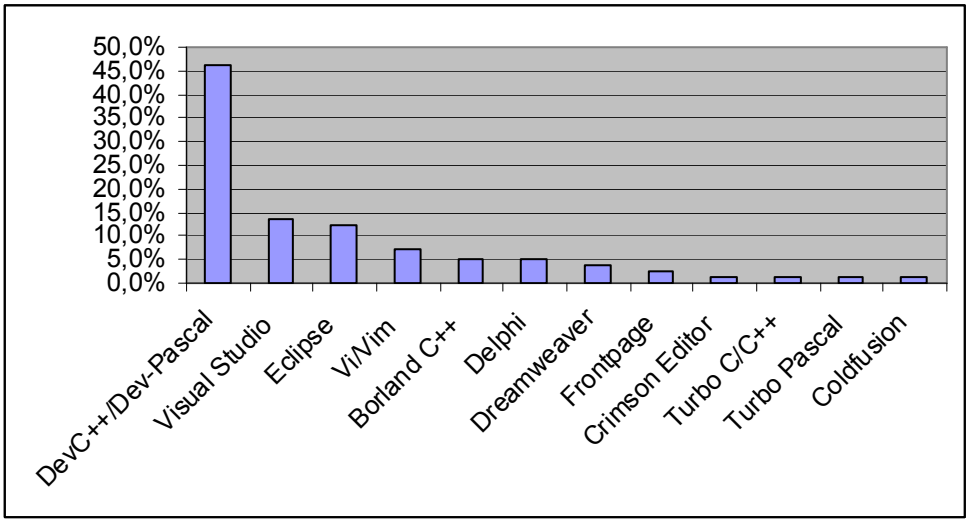


Figura 8 - IDEs habitualmente utilizadas pelos alunos (Turma B)

A Turma B teve quatro semanas disponíveis para a instalação e configuração da IDE Eclipse e da ferramenta *Hackystat*, ou seja, o dobro de tempo em comparação à Turma A. Os alunos utilizaram os mesmos Tutoriais de instalação e configuração de ferramentas disponibilizados para os alunos da Turma A. Tais Tutoriais são apresentados no Apêndice B. A Turma B desenvolveu cinco programas que fizeram parte do Estudo de Caso, também usando POO, a linguagem C++ e a IDE Eclipse.

4.3. Descrição do Estudo de Caso

O Estudo de Caso teve início logo após a conclusão da fase de planejamento descrita na Seção 3.3.1. Foi convocada uma reunião com os professores do Departamento de Computação da Universidade Federal de São Carlos que ministrariam aulas que envolvessem desenvolvimento de software no semestre em que o estudo foi realizado. O Estudo de Caso foi elaborado de forma a ter o mínimo impacto nas atividades normais das disciplinas participantes. Assim, ficou estabelecido que:

- A participação dos alunos seria voluntária;

- As atividades realizadas no contexto do Estudo de Caso não influenciaram na nota final da disciplina;
- O Professor da disciplina não teria que alterar o seu material de aula;
- O Professor da disciplina não teria que preparar novo material para o Estudo de Caso;
- Todo o suporte aos alunos para a instalação e configuração das ferramentas seria fornecido pelo pesquisador (autor deste trabalho);
- Os dados de PSP gerados pelos alunos seriam enviados diretamente ao pesquisador, e seriam tratados anonimamente;
- Seriam realizadas atividades dos Níveis 0 e 0.1 do PSP (*Baseline*);
- As atividades do PSP realizadas seriam modificadas de modo a terem o menor impacto nas atividades normais da disciplina;

Optou-se pela participação voluntária dos alunos para que a qualidade dos dados coletados não fosse comprometida, pois caso a participação fosse obrigatória, haveria a possibilidade dos alunos forneceram falsos dados. Pelos mesmos motivos, as atividades realizadas no contexto do Estudo de Caso não influenciavam na nota final da disciplina. Os dados coletados pelos alunos foram tratados de forma anônima e eram diretamente enviados para o pesquisador, para que os alunos não tivessem receio de que o Professor da disciplina pudesse usar tais dados para avaliar o seu desempenho. Assim, procurou-se extinguir os motivos para que os alunos fornecessem dados distorcidos com a realidade.

Dessa maneira, foi possível realizar o Estudo de Caso para analisar o impacto do PSP e de ferramentas de 3ª geração em disciplinas regulares do curso de Computação, e assim avaliar a possibilidade de aplicar o PSP em todas as disciplinas que envolvam desenvolvimento de software, de forma a incluir o PSP na grade curricular dos cursos de Computação. Para isso, é essencial que a aplicação do PSP seja invisível nas disciplinas pois, caso contrário, o PSP pode atrapalhar o aprendizado dos alunos quanto aos conceitos relativos à disciplina.

Salienta-se aqui que, consciente de que a implantação do PSP deve ser invisível, algumas decisões importantes foram tomadas para a realização deste Estudo de Caso, Neste caso, por exemplo, pelo fato da ferramenta não automatizar certas tarefas como o registro de em qual fase o defeito foi injetado, em qual fase foi removido e quanto

tempo se levou para remover tal defeito, o registro de defeitos foi simplificado, passando-se, então, a registrar somente o número total de defeitos. Essa medida foi tomada para se evitar que o processo ficasse muito trabalhoso, e é semelhante àquelas relatadas por Börstler et al. [Börstler et. al., 2002], comentada no Capítulo 2, de que uma versão mais leve do PSP permitiu o seu uso de forma transparente em cursos na *Umea University*.

A primeira etapa executada pelos alunos que optaram por participar do Estudo de Caso foi registrar-se no servidor público *Hackystat*, disponibilizado pela Universidade do Havaí. Essa etapa é muito simples, pois basta acessar uma página Web e fornecer um endereço de email, para o qual o servidor encaminha um email contendo usuário e senha.

A segunda etapa foi a instalação e configuração da IDE Eclipse e da ferramenta *Hackystat*. Essa etapa é executada uma única vez, pois estando as ferramentas instaladas no computador do aluno, elas podem ser utilizadas em todos os projetos daquele ponto em diante. Essa etapa foi especialmente difícil para a Turma A, pois os alunos não estavam habituados a usar a IDE Eclipse e, portanto, sua instalação e configuração se revelaram um aprendizado. Essa turma teve apenas duas semanas para instalar e configurar as ferramentas envolvidas no Estudo de Caso. Como os alunos já tinham que efetuar outras atividades, como estudar os conceitos de POO e C++, além de efetuar atividades de outras disciplinas, surgiram alguns problemas com a instalação, configuração e o uso das ferramentas. A Turma B teve um índice de sucesso maior nessa etapa. Mais detalhes dos problemas encontrados são discutidos na Seção 4.5.

Uma vez instaladas as ferramentas, para cada projeto a ser desenvolvido pelo aluno, a ferramenta *Hackystat* requer uma configuração bastante trivial, bastando ao aluno acessar uma página Web e fornecer o nome do Projeto e o caminho de diretório da máquina local em que os arquivos do projeto seriam armazenados.

Ao início de cada projeto, o aluno deveria realizar estimativas sobre o projeto a ser desenvolvido, como estimativa do número de linhas e estimativa do total de minutos necessários para completar o projeto. Para armazenar tais estimativas, foi fornecida uma planilha eletrônica, denominada *Base Histórica*, compatível com Microsoft Excel®. A escolha de uso de uma planilha eletrônica se deu pelos seguintes motivos:

- A ferramenta Hackystat não dá suporte direto e de fácil uso para o armazenamento e consulta de tais dados;
- Evitar a instalação de mais software no computador dos alunos;
- Permitir que os alunos acessassem rapidamente os dados, sem a necessidade de acesso à Internet;
- Agrupar os dados de todos os projetos em um único lugar de tal maneira que, ao realizar estimativas, os dados dos projetos anteriores estariam imediatamente disponíveis;

A partir desse ponto, o aluno desenvolve o software normalmente, como se não estivesse participando de um Estudo de Caso ou usando o PSP. A ferramenta *Hackystat* coleta os dados automaticamente. Ao final do desenvolvimento do software, o aluno entra na fase de Análise Final, e assim deve acessar uma página Web para visualizar os dados de seu desenvolvimento. O aluno, então, insere os seguintes dados na Planilha de Base Histórica:

- LOC Real: Número de linhas de código efetivamente criadas para a construção do software;
- LOC Comentários: Número de linhas de código que contêm comentários;
- Minutos Reais: Número total de minutos efetivamente consumidos para construir o software;
- Defeitos Graves Removidos: número de defeitos que foram difíceis de remover;
- Tipos de Defeitos Comuns: lista descritiva contendo os defeitos que mais se repetiram;
- Palavras-Chave: palavras que descrevem o projeto;

O número de defeitos graves removidos, os tipos de defeitos comuns e as palavras-chave não são coletados automaticamente pela ferramenta, mas sim fornecidos manualmente pelos alunos. Ao entrar com esses dados na Planilha de Base Histórica, os seguintes cálculos são realizados para o projeto:

a. Total LOC: Soma de LOC Real e LOC Comentários (Tamanho do Software);
b. Percentagem Comentário / LOC: Indica qual a percentagem de linhas que contêm comentários (Documentação);

c. Erro LOC: Diferença percentual do LOC estimado e do LOC efetivamente registrado ao final do projeto (Imprecisão de Planejamento);
d. Erro Minutos: Diferença percentual do tempo estimado e do tempo de desenvolvimento efetivamente registrado ao final do projeto (Imprecisão de Planejamento);
e. LOC/Hora: A quantidade média de LOC produzidas por hora (Produtividade);
f. Defeitos/KLOC: Número de defeitos graves registrados por mil linhas de código (Qualidade);
g. Defeitos/Hora: Número de defeitos graves registrados por hora (Qualidade);

Os cálculos realizados pela Planilha de Base Histórica permitem ao aluno compreender como ocorreu o desenvolvimento do software em relação às estimativas realizadas na fase de Planejamento do PSP, e também algumas de suas características de desenvolvimento. Assim, é desejável que o aluno minimize os cálculos apresentados nos itens C, D e F. Por outro lado, é interessante que o aluno maximize os cálculos B e E. Como os Tipos Comuns de Defeitos são descritivos, nenhum cálculo é efetuado com esses dados. A função desse dado é constituir um *checklist* básico para que o aluno possa identificar os erros que mais freqüentemente comete, para que em próximos projetos o aluno tenha maior cuidado e tente evitar repetir tais defeitos.

Após a fase de Análise Final, o aluno enviava os dados diretamente para o pesquisador, para que uma análise geral da turma fosse efetuada. Na entrega, os alunos podiam fornecer comentários para que se pudesse identificar quaisquer problemas ou pontos positivos não esperados. Ao final do Estudo de Caso, a Turma A entregou dados de três projetos, enquanto a Turma B efetuou a entrega de cinco projetos. A análise de resultados é apresentada na Seção 4.4.

Os resultados foram coletados através de questionários preenchidos anonimamente pelos alunos e, assim, avaliou-se o PSP através de dados indiretos. Evitou-se avaliar o PSP através de métricas do próprio PSP, como produtividade (KLOC/Hora) e densidade de defeitos (Defeitos/KLOC), para não permitir que a análise fosse distorcida, como relatado em outros trabalhos da área [Johnson & Disney, 1998].

O questionário respondido pelos alunos tinha duas seções: a primeira era voltada para todos os alunos da turma, e a segunda seção era voltada somente para os alunos

que participaram do Estudo de Caso. Essa divisão existe para que se avalie o impacto do Estudo de Caso tanto nos alunos que participaram no Estudo de Caso, quanto na turma como um todo. No caso dos alunos que não participaram, procurou-se averiguar o motivo da não participação.

4.4. Resultados

Os resultados apresentados nesta Seção são relacionados ao planejamento elaborado na Seção 3.3, ao contexto exposto na Seção 4.2 e à descrição do Estudo de Caso apresentada na Seção 4.3.

Análise 1 – Participação no Estudo de Caso. População: Todos os alunos.

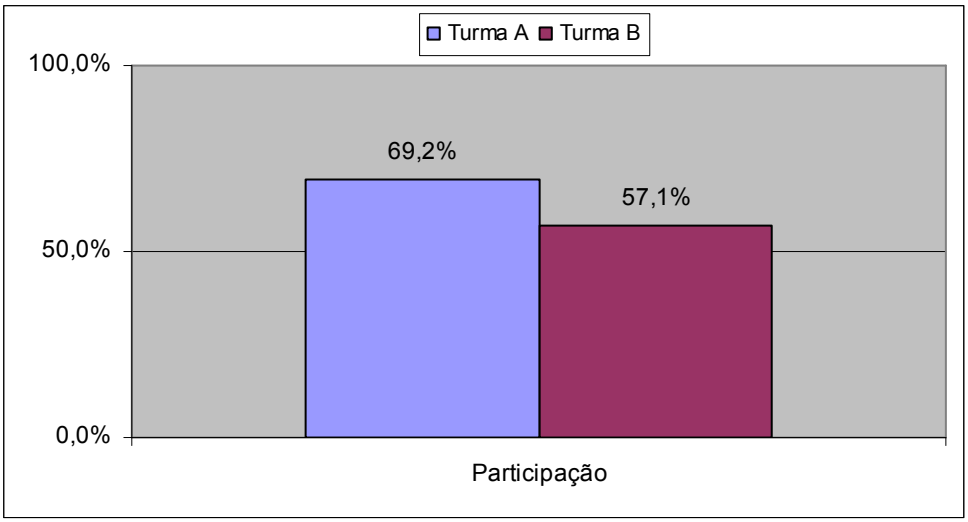


Figura 9 – Índice de participação no Estudo de Caso

A Turma A, apesar de ter apresentado mais problemas de instalação da ferramenta do que a Turma B, registrou uma participação maior no Estudo de Caso. Este dado sustenta a hipótese de que alunos mais experientes são menos receptivos às mudanças em seu ambiente de desenvolvimento. No entanto, não foi possível medir com precisão o nível de abandono dos alunos no Estudo de Caso, mas um levantamento informal junto aos alunos mostra indícios de que a taxa de abandono da Turma A foi muito alta, enquanto que na da Turma B foi pequena, com exceção da última atividade prevista que apresentou uma grande queda no número de alunos participantes.

Análise 2 – Alegações dos alunos para a não participação no Estudo de Caso. População: Alunos que não participaram do Estudo de Caso.

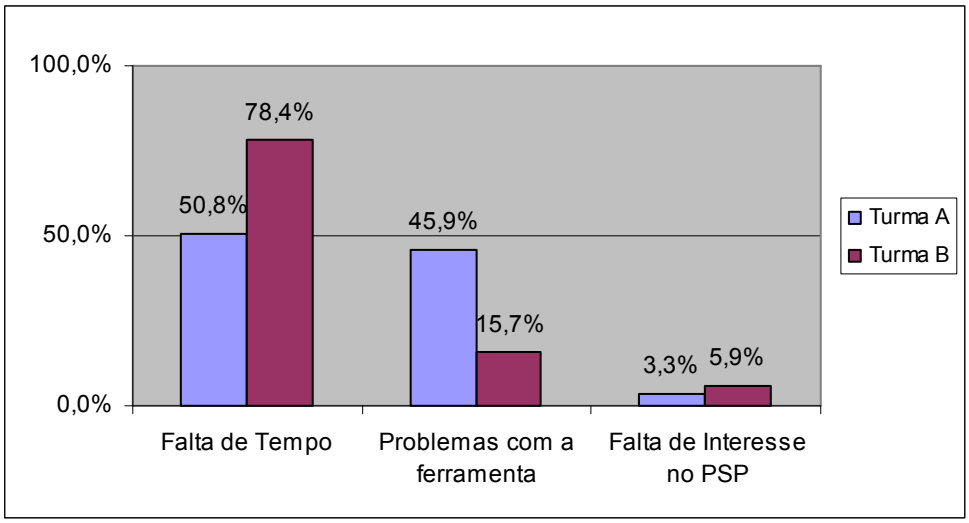


Figura 10 - Alegações dos alunos para não participarem do Estudo de Caso

A Figura 10 mostra que a falta de tempo foi o motivo mais citado pelos alunos de ambas as turmas para a não participação no Estudo de Caso. No entanto, a Turma A teve um índice de não participação relacionado a problemas com as ferramentas também alto. Este é provavelmente um sintoma do menor tempo que a Turma A teve disponível para instalar e configurar a ferramenta. Um número muito pequeno de alunos não participou do Estudo de Caso por não ter interesse no PSP.

Análise 3 – Percentagem de alunos que consideram processos de desenvolvimento de software, como PSP e CMM, importantes para a sua formação acadêmica. População: Todos os alunos.

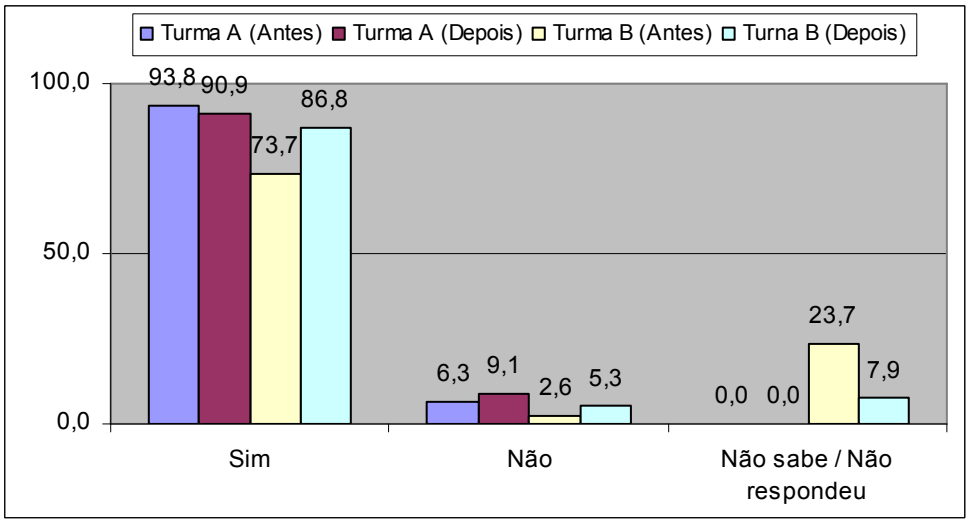


Figura 11 – Opinião dos alunos sobre a importância de processos de desenvolvimento de software

É possível notar na Figura 11 que, inicialmente, mais alunos da Turma A do que da Turma B consideravam processos de desenvolvimento de software importantes para o seu desenvolvimento acadêmico. No entanto, ao final da disciplina (e do Estudo de Caso), enquanto houve uma pequena queda no número de alunos da Turma A que consideravam importantes tais processos, ocorreu um considerável aumento no número de alunos da Turma B com tal opinião.

Algumas hipóteses podem ser formuladas a partir dos dados e do contexto do Estudo de Caso. Alunos com mais experiência aparentam dar uma menor importância para processos de desenvolvimento de software, e a evolução da Turma B pode estar ligada a quatro fatores: maior experiência, menos problemas com as ferramentas do Estudo de Caso, desenvolvimento de um número maior de projetos no contexto do PSP e menor índice de abandono no Estudo de Caso. Embora no estudo de Lisack [Lisack, 1999] os alunos tenham usado ferramentas de primeira geração, a autora comenta que alunos com menos experiência têm maior dificuldade em seguir um processo sistemático como é o PSP, pois já estão ocupados se dedicando ao aprendizado de linguagens de programação. No caso deste estudo, a Turma A era menos experiente e talvez esse tenha sido outro fator que tenha levado ao decréscimo observado nessa turma.

Análise 4 – Percentagem de alunos que relataram interesse próprio em aprender processos de desenvolvimento de software, como PSP e CMM. População: Todos os alunos.

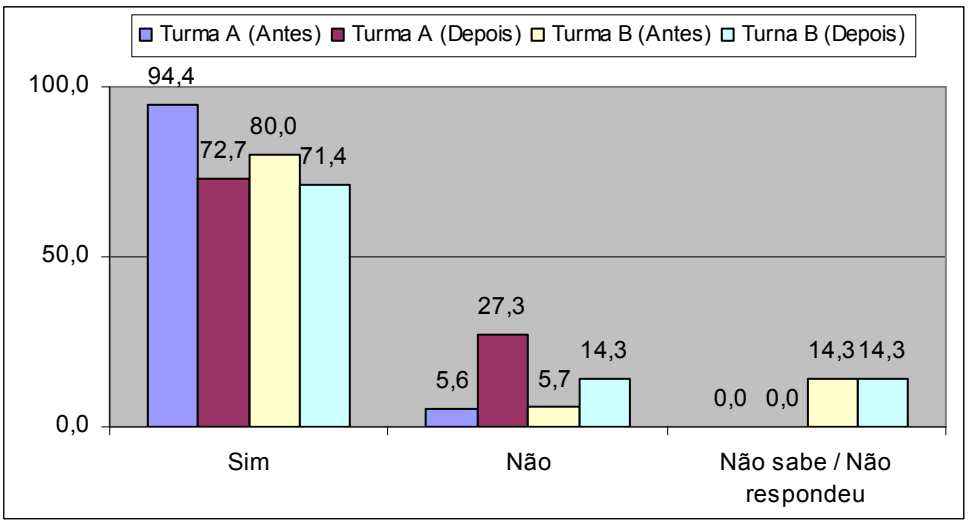


Figura 12 - Relato do interesse dos alunos em processos de desenvolvimento de software

A Figura 12 mostra que houve uma queda em ambas as turmas no interesse em processos de desenvolvimento de software em relação à opinião inicial, coletada antes do Estudo de Caso. Acredita-se que essa queda esteja relacionada à sobrecarga de trabalho gerada pelo uso do PSP e, principalmente, à instalação e configuração da ferramenta, daí uma queda mais acentuada na Turma A. Assim, os alunos podem ter associado que processos de desenvolvimento de software exigem muito trabalho para serem colocados em prática.

Esses dados mostram um aspecto negativo do Estudo de Caso, pois com o uso de ferramentas de 3ª geração esperava-se que o nível de interesse dos alunos se mantivesse o mesmo, ou então aumentasse.

Análise 5 – Relato de estimativas realizadas pelos alunos. População: Todos os alunos.

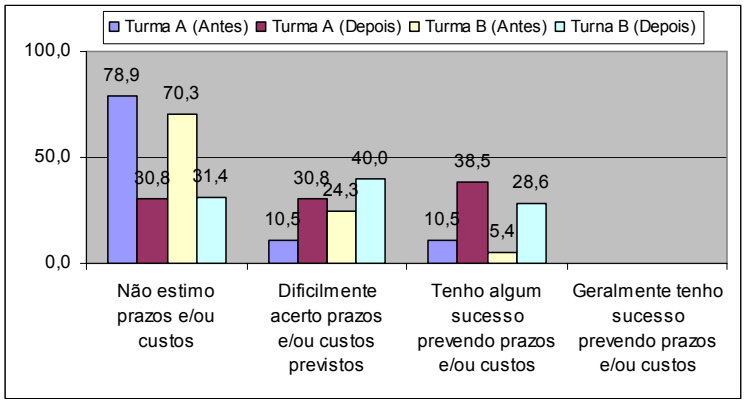


Figura 13 – Relato de estimativas dos alunos

A Figura 13 mostra que o Estudo de Caso teve um impacto positivo nos alunos em relação às estimativas realizadas. O comportamento das duas turmas é muito parecido, mostrando uma acentuada queda na percentagem de alunos que não realizam estimativas, ao mesmo tempo que há um aumento da percentagem de alunos que tem algum sucesso em prever prazos e custos. O índice de participação no Estudo de Caso parece não influenciar nesse aspecto, sugerindo que o que mais contribuiu para a mudança de postura dos alunos foram atividades que todos participaram, como as palestras e a leitura dos materiais disponibilizados sobre PSP.

Análise 6 – Conhecimento sobre os conceitos do PSP. População: Todos os alunos.

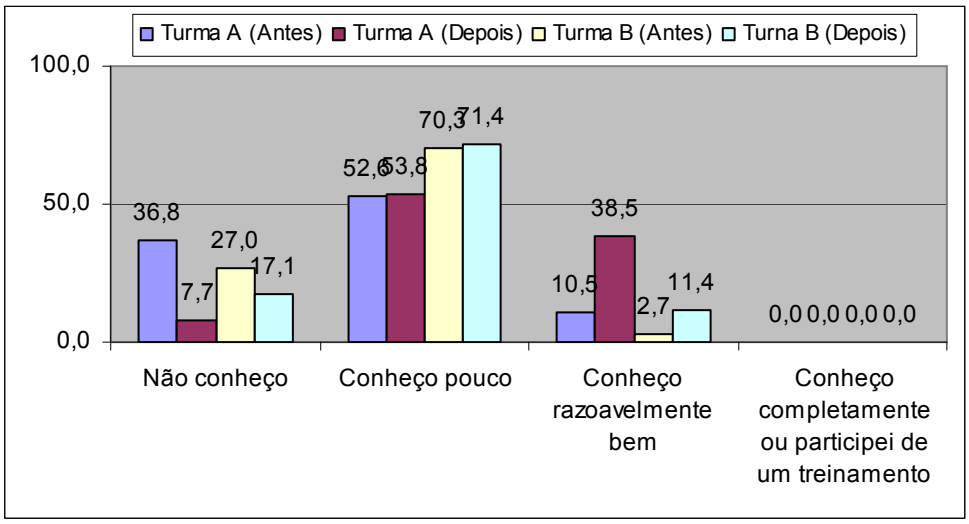


Figura 14 - Conhecimento dos alunos sobre os conceitos do PSP

A Figura 14 mostra que ao final do Estudo de Caso, a percentagem de alunos de ambas as turmas que dizem conhecer razoavelmente os conceitos do PSP aumentou, enquanto que o número de alunos que relataram que não conhecem diminuiu. O resultado foi melhor na Turma A do que na Turma B, o que pode ser atribuído ao maior índice de participação ou à menor resistência dos alunos da Turma A em aprender um novo processo de desenvolvimento de software.

Análise 7 – Relato dos alunos sobre o uso do PSP. População: Alunos que participaram do Estudo de Caso.

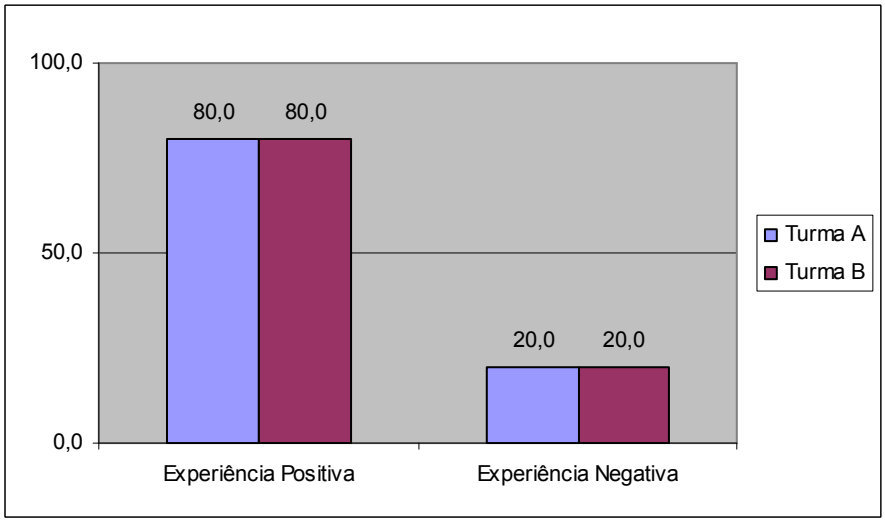


Figura 15 - Relato sobre o uso do PSP dos alunos que participaram do Estudo de Caso

A Figura 15 mostra que a maioria dos alunos que participou do Estudo de Caso relatou que o uso do PSP se mostrou uma experiência positiva, e que as Turmas A e B apresentaram exatamente as mesmas percentagens nessa análise. No entanto, conclusões sobre tais dados devem ser realizadas levando-se em conta também os dados da análise 8, a seguir.

Análise 8 – Opinião dos alunos sobre o uso espontâneo do PSP em projetos futuros. População: Alunos que participaram do Estudo de Caso.

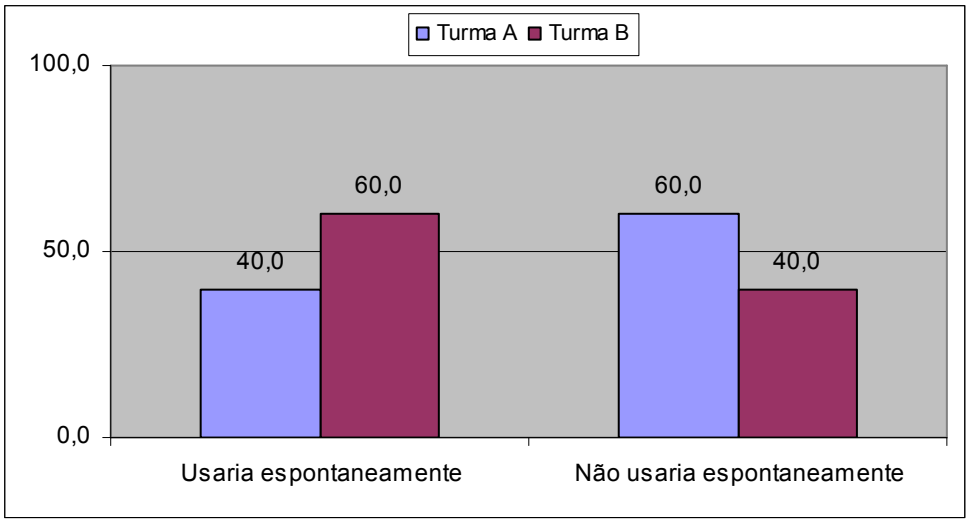


Figura 16 – Uso espontâneo do PSP no futuro

A Figura 16 mostra que a Turma A apresenta uma tendência contrária à tendência da Figura 15, ou seja, embora a maioria dos alunos tenha relatado que o contato com o PSP se mostrou uma experiência positiva, constatou-se uma queda na percentagem de alunos que usaria espontaneamente o PSP ao término do Estudo de Caso. Isso significa que a maioria dos alunos da Turma A que julgou o PSP uma experiência positiva, não voltaria a utilizá-lo espontaneamente. Esses dados são de muita importância, pois mostram que apesar dos alunos terem reconhecido vantagens ao usar o PSP, o trabalho extra para utilizá-lo parece não valer a pena do ponto de vista dos mesmos. Tal fato possivelmente está relacionado com o pouco tempo disponível para a Turma A instalar e configurar a ferramenta e o pequeno número de programas desenvolvidos nessa turma.

Por outro lado, dos alunos da Turma B que participaram do Estudo de Caso, 60% voltariam a utilizar o PSP espontaneamente. Mesmo que esse número não seja tão grande, é um indicativo de que alunos dessa turma julgaram que o trabalho considerado

extra do uso do PSP acabou tendo maior retorno do que considerado pela Turma A. Isso mostra que para a aplicação do PSP ser um sucesso os alunos devem considerar, simultaneamente, que a ferramenta de apoio ao método está facilitando o seu trabalho e também que o PSP está melhorando o seu trabalho em ao menos um aspecto. Assim, os dados da Figura 18 e da Figura 19 podem ser usados para a avaliação contínua da aplicação do PSP.

A Figura 17 mostra que quando esta análise inclui alunos que nunca usaram o PSP, o índice de alunos que usariam espontaneamente aumenta comparada à Figura 16, pois, como mostrado na Figura 12, o nível de interesse dos alunos que nunca usaram o PSP é alto.

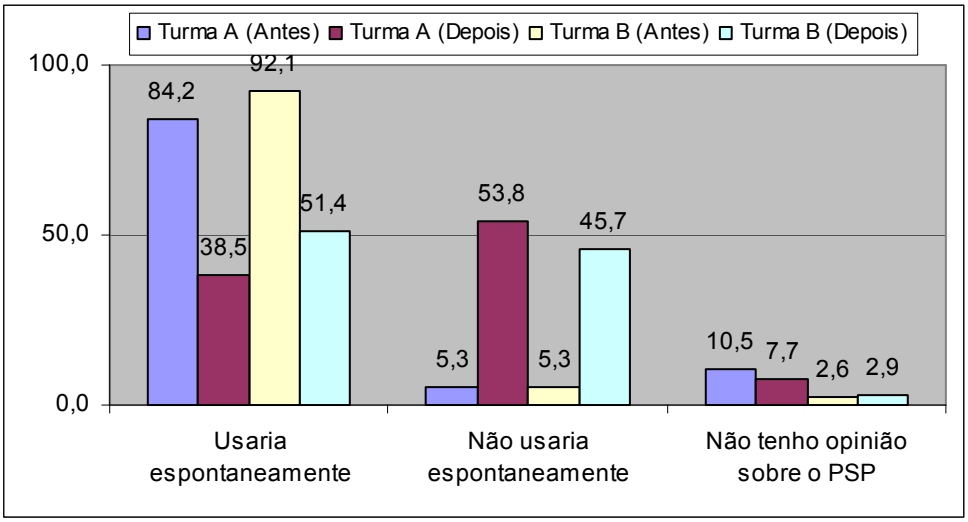


Figura 17 - Percentagem de alunos que voltariam a usar o PSP espontaneamente

Análise 9 – Análise do impacto do PSP no trabalho dos alunos. População: Alunos que participaram do Estudo de Caso.

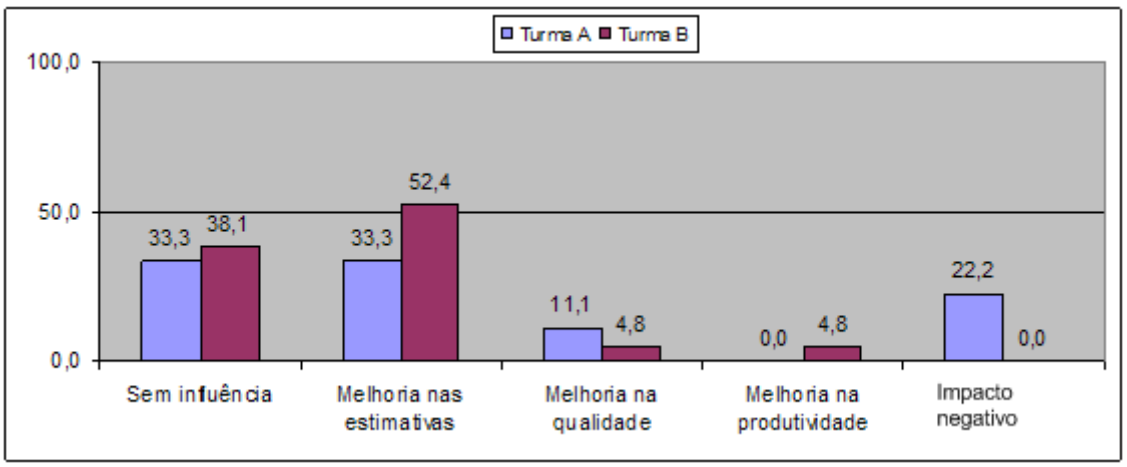


Figura 18 – Impacto do PSP no trabalho dos alunos

A Figura 18 mostra que um número muito pequeno de alunos relatou que houve melhoria na qualidade ou na produtividade de seu trabalho. Esses dados já eram esperados, uma vez que não se realizaram atividades dos níveis 1 e 2 do PSP. No entanto, um número considerável de alunos relatou que houve melhorias nas estimativas, o que está alinhado com o que mostram os dados da Figura 13. Esse é um indício de que mesmo atividades simples de Planejamento e Análise Final podem motivar os alunos a usarem o PSP. Por outro lado, um número grande ainda de alunos relatou que não houve influência alguma do PSP em seu trabalho, e preocupantes 22,2% dos alunos da Turma A relataram que houve um impacto negativo. Tal fato provavelmente está ligado aos problemas na instalação da ferramenta e ao fato dos alunos da Turma A não estarem habituados ao uso da IDE Eclipse e, portanto, foram obrigados a dedicar boa parte do tempo que normalmente utilizariam estudando os conceitos da disciplina em resolver problemas técnicos. Tal fato provavelmente também teve influências nos dados da Figura 19, apresentada na Análise 10, a seguir.

Análise 10 – Opinião dos alunos sobre o uso da Ferramenta *Hackystat*. População: Alunos que participaram do Estudo de Caso.

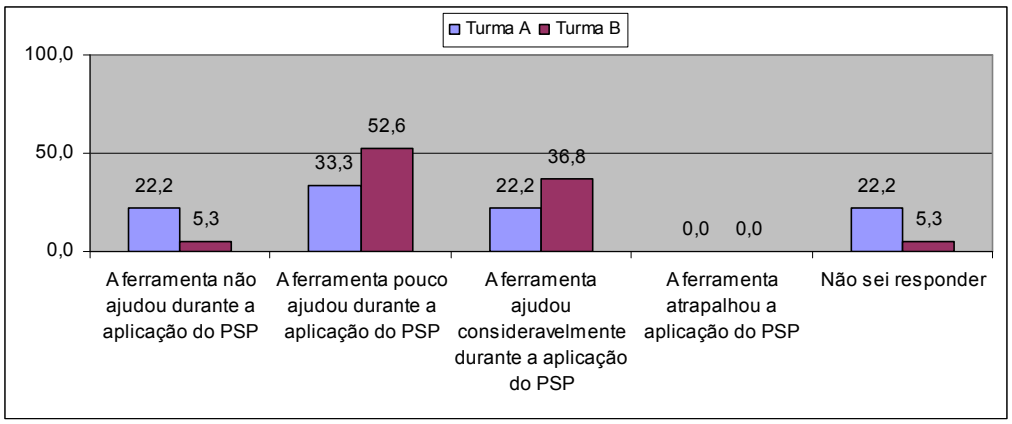


Figura 19 – Opinião dos alunos sobre a ferramenta *Hackystat*

A Figura 19 mostra que a Turma A tem uma opinião negativa sobre a ferramenta muito mais intensa do que alunos da Turma B, pois a percentagem de alunos que relataram que a ferramenta não ajudou na aplicação do PSP é de 22,2%, contra 5,3% na Turma B. Esta turma também apresenta melhores índices de alunos que julgaram que a ferramenta ajudou consideravelmente na aplicação do PSP. Provavelmente, este ainda é um sintoma dos problemas técnicos que foram muito mais freqüentes na Turma A. Nenhum aluno relatou que a ferramenta atrapalhou a aplicação do PSP.

Análise 11 – Maiores dificuldades relatadas pelos alunos durante a participação no Estudo de Caso. População: Alunos que participaram do Estudo de Caso.

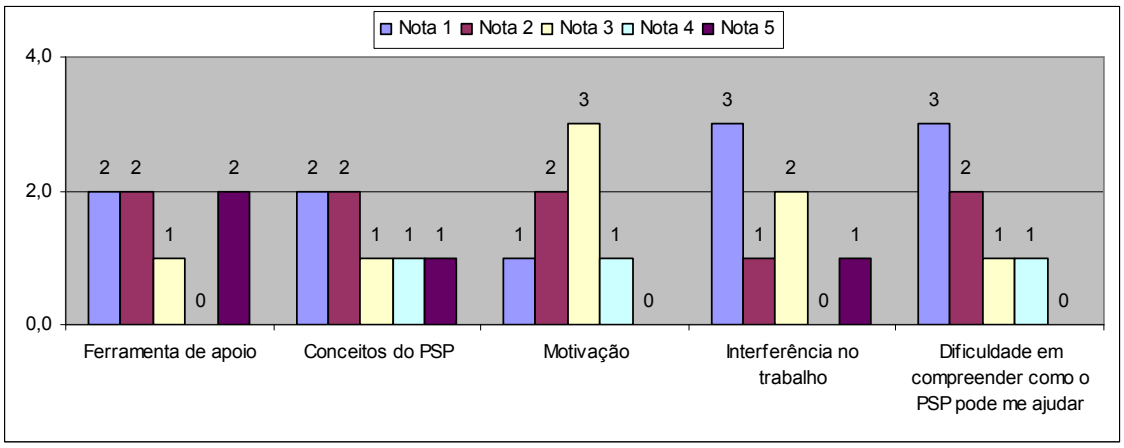


Figura 20 – Maiores dificuldades dos alunos da Turma A no Estudo de Caso

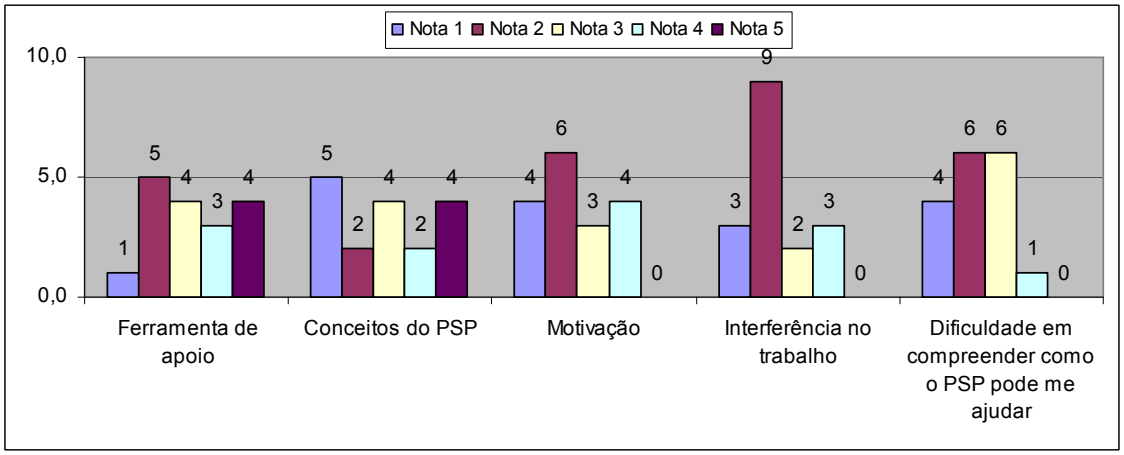


Figura 21 - Maiores dificuldades dos alunos da Turma B no Estudo de Caso

As Figuras 20 e 21 mostram, respectivamente, as principais dificuldades relatadas pelos alunos no Estudo de Caso das turmas A e B. Se normalizarmos os dados de acordo com os pesos das notas, obtemos a Figura 22.

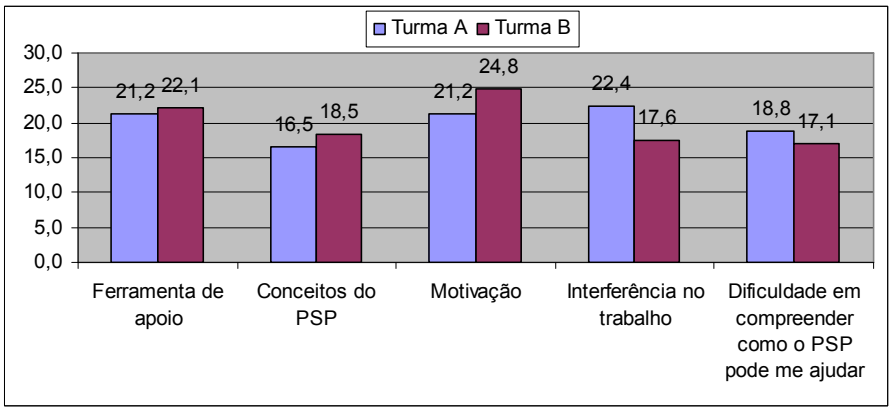


Figura 22 - Normalização das maiores dificuldades dos alunos

A Figura 22 mostra que para a Turma A, a maior dificuldade no Estudo de Caso foi a interferência nas atividades da disciplina, com 22,4% das notas distribuídas. Em seguida e muito próximos, vieram problemas com ferramentas de apoio e falta de motivação com 21,2%. Problemas relacionados com os conceitos do PSP e a dificuldade em compreender como o PSP pode ajudá-los foram os motivos menos citados pelos alunos da Turma A.

A Turma B, por outro lado, apresenta dados um pouco diferentes. O problema mais citado foi a falta de motivação em usar o PSP com 24,8% das notas distribuídas. Em seguida, como ocorreu na Turma A, o problema mais citado foram os problemas relacionados à instalação, configuração e uso da ferramenta de apoio com 22,1%. Ao contrário da Turma A, em que a interferência nas atividades da disciplina foi o maior problema, na Turma B tal interferência recebeu apenas 17,6% das notas distribuídas.

No caso da Turma A, é provável que as dificuldades encontradas com a instalação e configuração do Eclipse e da *Hackystat* tenham refletido no alto índice de interferência nas atividades da disciplina. Conseqüentemente, a motivação para participar do Estudo de Caso pode ter diminuído consideravelmente.

A Turma B também relatou que enfrentou problemas técnicos, mas essa turma teve o dobro de tempo para solucionar tais problemas em relação à Turma A. Provavelmente essa seja a causa do baixo índice de reclamações sobre a interferência nas atividades da disciplina. No entanto, a falta de motivação foi o principal problema relatado por essa turma, sustentando o relato dos estudos que indicam que alunos mais experientes são mais resistentes a novos métodos de desenvolvimento [Runson, 2001].

4.5. Identificação das dificuldades e dos benefícios

As primeiras dificuldades foram encontradas antes mesmo do início do Estudo de Caso, e eram relativas à compatibilidade da ferramenta *Hackystat* com o ambiente de programação dos alunos. Do total de nove turmas que realizariam atividades de programação naquele semestre, somente duas puderam participar do Estudo de Caso.

Das nove turmas disponíveis, quatro turmas eram da disciplina “Laboratório de Compiladores”, todas as quatro turmas tendo um mesmo professor. Nessa disciplina, os alunos desenvolveriam dois projetos em Java, havendo total liberdade para a escolha do

ambiente de programação. Embora o Eclipse fosse uma opção compatível com a disciplina, todos os alunos optaram por usar o ambiente de programação *NetBeans*™, por já terem utilizado essa IDE em disciplinas anteriores. Não há sensores *Hackystat* compatíveis com a IDE *Netbeans*™.

Outras duas turmas eram da disciplina "Programação de Computadores", ambas orientadas por um mesmo professor. Nessa disciplina os alunos desenvolveriam entre três e seis programas na linguagem C++ utilizando o ambiente *DevC++*. No entanto, também não há sensores *Hackystat* compatíveis com o ambiente *DevC++* e, assim, novamente não se pode aplicar o Estudo de Caso.

Uma turma da disciplina "Organização e Recuperação da Informação" também não pôde participar do Estudo de Caso por incompatibilidade com a ferramenta *Hackystat*. Nessa disciplina, os alunos desenvolvem cerca de três programas usando a linguagem C++, tendo como opções de ambientes de programação o *DevC++* ou o *Microsoft*™ *Visual Studio*™. No caso, existe um sensor compatível com o ambiente Visual Studio, mas não foi possível instalá-lo pois tal sensor requer que o *Microsoft .Net Framework* versão 1 esteja instalado na máquina do desenvolvedor. As máquinas dos laboratórios do Departamento de Computação têm instaladas as versões 2 ou 3 do Microsoft .Net Framework, e o sensor *Hackystat* para *Visual Studio* é incompatível com essas versões. Reverter a versão do *.Net Framework* para a versão 1 traria muitos problemas de compatibilidade com várias ferramentas e, portanto, essa opção foi descartada. Assim, embora exista o sensor compatível com *Visual Studio*, não foi possível utilizar essa *IDE* por conta dos problemas técnicos relatados.

Fica claro, portanto, que a instalação e a configuração das ferramentas de apoio podem ser problemáticas e, portanto, acabam dificultando a aplicação do PSP, pois faz com que os alunos tenham que abdicar de parte de seu tempo de estudo para que se dediquem à instalação das ferramentas. No entanto, esse é um problema que tende a ocorrer uma única vez, pois uma vez instalada e configurada corretamente, a ferramenta é quase imperceptível aos alunos. Pode-se concluir, portanto, que o trabalho da instalação e configuração das ferramentas só se caracteriza como negativa quando as ferramentas são utilizadas por um curto período de tempo. Quando se usam as ferramentas por um período de tempo maior, o trabalho de instalação acaba sendo "diluído", pois como as ferramentas são mais utilizadas, os benefícios (uso das

ferramentas) tendem a superar os custos (instalação das ferramentas). Para "diluir" o custo de uso de ferramentas de 3ª geração, deve-se minimizar o número de instalações e, ao mesmo tempo, maximizar o uso da ferramenta, através das seguintes ações:

- Evitar utilizar mais de um computador no desenvolvimento dos projetos (ex: usar um computador no laboratório, um em casa e outro no trabalho implica em triplicar o custo de instalação);
- Em ambiente acadêmico é desejável que os professores utilizem o mesmo ambiente de programação para todas as disciplinas, pois se cada professor utilizar um ambiente diferente os alunos terão que instalar diferentes sensores para diferentes disciplinas;
- Em ambiente acadêmico, é desejável que a ferramenta seja instalada o quanto antes pelos alunos, de preferência na primeira disciplina cursada em que há desenvolvimento de software. Assim, uma vez instalada a ferramenta a mesma poderá ser utilizada em um número maior de disciplinas posteriores.

Portanto, somente duas turmas puderam participar do Estudo de Caso, que são as turmas descritas na Seção 4.2. No entanto, mais problemas técnicos surgiram com essas turmas. Como descrito na Seção 4.3, optou-se por utilizar o servidor *Hackystat* público disponibilizado pela Universidade do Havaí. Portanto, era necessário que os sensores instalados nas máquinas dos alunos tivessem acesso à Internet para enviar os dados para o servidor. Mas a combinação dos softwares *firewall* e *proxy* instalados na rede do Departamento de Computação impedia a comunicação entre os sensores e o servidor público. A ferramenta *Hackystat* possui configuração de *proxy*, mas mesmo assim a comunicação não era estabelecida. Para solucionar o problema, foi preciso requisitar para o administrador de redes do departamento para que ele abrisse uma exceção nas regras de *firewall* para que fosse possível a comunicação entre sensores e servidor. Esse problema atrasou o cronograma do Estudo de Caso em alguns dias por ser de difícil diagnóstico, uma vez que os laboratórios da pós-graduação (onde muitos dos testes eram realizados) não apresentavam tal problema, que era reproduzido exclusivamente no laboratório dos alunos de graduação.

Outro problema identificado foi que alunos desenvolvem seus projetos ora nos laboratórios do Departamento, ora em suas residências. Isso fazia com que o número de instalações e configurações dobrasse em alguns casos. Alguns alunos não possuem

acesso à Internet em suas residências, tornando impossível a comunicação direta com o servidor. Uma solução técnica para esse problema foi apresentada, em que os dados podiam ser copiados para um meio de armazenamento temporário (como um *pen drive*) e então copiados na máquina do laboratório para os dados serem, enfim, transferidos para o servidor. No entanto, os alunos achavam esse procedimento muito complicado.

Foi identificado um problema técnico específico em um dos laboratórios disponíveis para os alunos da graduação. Nesse laboratório as máquinas não possuíam um usuário e senha para cada um dos alunos, ou seja, todos os alunos compartilhavam de um mesmo "Usuário Administrador". Em um laboratório com essas características, fica impossível instalar a *Hackystat*, pois não há como separar os dados coletados pela ferramenta para cada um dos alunos. É necessário que computadores compartilhados por mais de um desenvolvedor tenham contas de usuários separados. Chegou-se a elaborar uma solução técnica que solucionasse o problema, mas a solução mostrou-se muito complexa de ser utilizada e muito propensa a falhas.

Outra dificuldade identificada foi que alguns recursos da ferramenta *Hackystat* que eram compatíveis com o PSP não puderam ser utilizados, pois o seu uso significaria uma sobrecarga muito grande nos alunos. Esse é o caso do Sensor compatível com a ferramenta Ant, que caso utilizado possibilitaria que mais atividades pudessem ser totalmente automatizadas, como o envio do número de linhas de código para o servidor *Hackystat*. Se o sensor compatível com Ant fosse utilizado, seria então possível também utilizar sensores compatíveis com testes unitários (*JUnit* ou *CPPUnit*) que são capazes de enviar relatórios de número de defeitos injetados e removidos.

Essas duas atividades (contagem de número de linhas e registro de defeitos) seriam extremamente úteis de serem automatizas para o Estudo de Caso, mas a instalação das ferramentas necessárias e dos sensores compatíveis é extremamente complexa para os alunos. Além das dificuldades técnicas, os alunos não conheciam o conceito de testes unitários, e provavelmente não saberiam construir casos de testes úteis. Assim, optou-se por não utilizar tais recursos, uma vez que se os alunos fossem obrigados a utilizá-los, o Estudo de Caso estaria sendo colocado em risco com o aumento de desistência dos alunos em participar do Estudo de Caso.

Através do *feedback* de alguns alunos, constatou-se que os alunos se sentiram levemente sobrecarregados em preencher a Planilha Eletrônica de Dados Históricos manualmente, sendo que alguns sugeriram que os dados fossem passados automaticamente. Embora tal solução técnica fosse impossível de ser elaborada para o Estudo de Caso, este *feedback* dos alunos endossa que as atividades do PSP devem ser automatizadas sempre que possível, pois mesmo atividades simples e rápidas como o preenchimento de alguns poucos dados em uma planilha eletrônica podem ser consideradas um distúrbio nas atividades normais do desenvolvedor.

Tanto na palestra inicial sobre PSP e ferramentas de apoio, quanto no material de apoio fornecido aos alunos, deixou-se claro que a participação no Estudo de Caso era voluntária, e que o Estudo de Caso não influenciava, de maneira alguma, na nota da disciplina. No entanto, ainda assim alguns alunos se sentiram obrigados a realizar as atividades do PSP, e o fizeram com o único intuito de serem recompensados com notas, apesar dos esforços do Estudo de Caso em coletar os dados anonimamente. Este é um aspecto surpreendente do Estudo de Caso, que deverá ser averiguado em maiores detalhes em trabalhos futuros.

Apesar das dificuldades identificadas, também se observaram alguns benefícios durante o Estudo de Caso. A ferramenta, uma vez instalada e configurada corretamente, quase não requer a intervenção do programador para coletar medidas de tempo. Alunos que tiveram sucesso em instalar rapidamente a ferramenta e que estavam acostumados a usar a IDE Eclipse relataram que não houve interferência no trabalho normal da disciplina. Este é um ponto positivo, pois as ferramentas de 3ª geração se propõem exatamente a modificar minimamente as atividades normalmente executadas pelo desenvolvedor.

Outro ponto positivo identificado no Estudo de Caso é que alunos que relataram entender bem o PSP, em geral, querem voltar a utilizar o PSP, mesmo que tenham tido problemas com as ferramentas. Alunos que desenvolveram mais do que três programas utilizando o PSP têm uma postura muito mais positiva em relação ao uso do PSP do que os outros alunos, chegando a comentar que o seu uso foi tão positivo que o PSP deveria ser implantado em todas as disciplinas de programação do Departamento de Computação.

Por fim, não foi possível seguir completamente o planejamento apresentado na Seção 3.3.1 devido a limitações de tempo, das ferramentas de apoio utilizadas e do pequeno número de participantes do Estudo de Caso. A Tabela 24 resume as métricas que foram utilizadas no estudo de caso e as métricas que, por algum dos motivos citados, não puderam ser coletadas junto aos alunos. Dessa maneira, pretende-se que trabalhos futuros complementem o Estudo de Caso aqui apresentado para que todo o planejamento seja cumprido. Trabalhos futuros são discutidos na Seção 6.2.

Tabela 24 – Métricas envolvidas no estudo de caso

Métrica	Fase de desenvolvimento	Tipo de coleta
Estimativa de linhas de código	Planejamento	Manual
Total de linhas de código	Análise Final	Automático
Estimativa de Tempo	Planejamento	Manual
Tempo efetivamente necessário para completar a tarefa	Análise Final	Automático
Número de linhas de comentário	Análise Final	Automático
Total de defeitos injetados	Análise Final	Manual
Tempo em cada uma das fases de desenvolvimento	Não coletado	
Defeitos injetados em cada fase de desenvolvimento		
Número de linhas de código reutilizadas		

4.6. Considerações Finais

Este capítulo apresentou o Estudo de Caso realizado em Ambiente Acadêmico. O Estudo de Caso foi executado paralelamente em duas turmas de disciplinas de programação do Departamento de Computação da Universidade Federal de São Carlos. Mais turmas foram convidadas a participar, mas problemas técnicos impediram que isso se concretizasse.

Na Seção 4.2 foi apresentado o contexto do Estudo de Caso e a caracterização das turmas A e B. A Turma B se mostrou mais experiente em programação do que a Turma A e, junto com diferenças quanto ao tempo disponível para instalar e configurar o conjunto de ferramentas necessárias, isso teve grande impacto nos resultados das turmas.

A descrição do Estudo de Caso foi apresentada na Seção 4.3, e os resultados de ambas as turmas apresentados na Seção 4.4. Os resultados mostram que os alunos têm dificuldades para instalar e configurar todas as ferramentas necessárias, mas que uma vez que isso foi realizado, a sobrecarga gerada pelo PSP e pelas ferramentas passa a ser mínima.

A Seção 4.5 reúne os benefícios do uso do PSP e das ferramentas de apoio registrados no Estudo de Caso, e as inúmeras dificuldades encontradas. Algumas das dificuldades registradas eram esperadas, no entanto a aplicação permitiu que se identificassem dificuldades não esperadas, e essa informação é muito valiosa no planejamento de futuros Estudos de Caso.

No próximo capítulo é apresentado o Estudo de Caso em um ambiente de pequena empresa, no qual outros aspectos do uso de ferramentas de 3ª geração no apoio ao PSP puderam ser verificados.

CAPÍTULO 5

ESTUDO DE CASO 2: AMBIENTE DE PEQUENA EMPRESA

5.1. Considerações Iniciais

Neste capítulo apresenta-se o segundo Estudo de Caso deste trabalho, elaborado em um ambiente de pequena empresa. Este estudo possui algumas características diferentes do estudo elaborado em ambiente acadêmico, descrito no Capítulo 4, embora também existam entre eles pontos em comum. O planejamento deste Estudo de Caso foi apresentado na Seção 3.3.2.

A pequena empresa em questão é a *Linkway Internet & Telecom*, que é um pequeno provedor de Internet que está há dez anos no mercado de desenvolvimento de 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. Há bastante tempo essa empresa está empenhada 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 e, mais recentemente, com a proposta do MPS.BR, pelo fato desse modelo ser mais adequado às empresas de pequeno porte.

O contexto do Estudo de Caso elaborado, alguns dados da empresa e a caracterização dos participantes são apresentados na Seção 5.2. As descrições do Estudo de Caso e das atividades realizadas são apresentadas na Seção 5.3, enquanto os resultados coletados são analisados e apresentados na Seção 5.4. Na Seção 5.5 comentam-se os benefícios e as dificuldades identificadas durante o Estudo de Caso, e são discutidas modificações na ferramenta e no PSP que reduziriam o número de dificuldades identificadas. Por fim, a Seção 5.6 apresenta as considerações finais.

5.2. Contexto do Estudo de Caso

O Estudo de Caso presente neste capítulo foi elaborado em uma pequena empresa de desenvolvimento de software situada na cidade de São Carlos. Essa empresa já utilizava o PSP há dois anos, com o apoio da ferramenta de 2ª geração *Process Dashboard*. Como mencionado, os principais produtos da empresa são *sites* de Internet personalizados, desenvolvidos utilizando a linguagem de programação Java [Sun, 2007] e as tecnologias EJB3[Sun, 2007c] e JSF [Sun, 2007b].

A empresa possui três programadores Java que utilizam o PSP Nível 1.1 (Planejamento). Os dados do PSP dos desenvolvedores são, inclusive, utilizados no contexto de uma estratégia de planejamento e acompanhamento de projetos [Sanchez et. al., 2007], cuja base principal reside na avaliação e melhoria contínua fornecida pelo PSP. O gerente da área de desenvolvimento de software da *Linkway* apóia o uso do PSP, que é um dos fatores-chave para o sucesso da aplicação em empresas, como discutido no Capítulo 2.

No entanto, quando o Estudo de Caso foi realizado, um dos programadores da empresa estava em férias e, portanto, não pôde participar. Como será descrito em maiores detalhes na Seção 5.4, houve um problema técnico na configuração da ferramenta *Hackystat* no ambiente de desenvolvimento de outro programador. Assim, apenas um único programador da empresa pôde participar deste estudo. Isso limitou bastante o escopo do estudo e, por haver somente um participante, os resultados não são viáveis de serem generalizados. Assim, optou-se por analisar os resultados sob um ponto de vista diferente do Estudo de Caso em Ambiente Acadêmico, focando então em analisar as diferentes métricas coletadas por ferramentas de diferentes gerações.

O programador que participou do Estudo de Caso respondeu ao mesmo questionário de caracterização que os alunos do meio acadêmico responderam no Estudo de Caso 1. Assim, o programador caracteriza-se por ter muita experiência em meio acadêmico, uma vez que este já concluiu o curso de graduação em Informática. Na indústria de software, o programador possui pouca experiência, tendo trabalhado menos de um ano à época do Estudo de Caso.

Os ambientes de programação que o desenvolvedor habitualmente utiliza são o *Eclipse*, o *Netbeans* e o *Dreamweaver*. Estes dois últimos não possuem sensores

compatíveis com a ferramenta *Hackystat*, e durante o Estudo de Caso somente a IDE *Eclipse* foi utilizada. O programador relatou ter algum sucesso prevendo prazos e/ou custos, e que para tal utiliza um processo bem definido (PSP). Também é importante salientar que o programador considera processos de melhoria contínua, como o PSP, importantes e interessantes, e que foi relatado que os conceitos do PSP são plenamente conhecidos.

De fato, é importante deixar claro que o programador acredita no PSP e atualmente o utiliza espontaneamente, mesmo que, em princípio, tenha sido obrigado a utilizar esse processo pela gerência. O desenvolvedor sempre utilizou o PSP com o apoio de uma ferramenta de 2ª geração, e considera que isso ajudou consideravelmente para a aplicação do PSP. Sendo esse programador extremamente comprometido com o PSP, é preciso ter cautela com as análises realizadas sobre os dados do Estudo de Caso, pois os mesmos podem refletir uma situação ótima de máxima motivação de uso do PSP, que é diferente da situação da maioria dos desenvolvedores de software. Assim, esse programador se esforça ao máximo para que as métricas coletadas pela ferramenta *Dashboard* (que requer que o desenvolvedor constantemente forneça dados de seu desenvolvimento) tenham a máxima qualidade e precisão.

Sobre o impacto do PSP em seu trabalho, o desenvolvedor relatou que houve melhoria somente no planejamento dos projetos e nas estimativas realizadas, o que é compatível e esperado do Nível 1.1 do PSP. A principal dificuldade relatada pelo programador no uso do PSP foi a interferência no trabalho, o que é uma característica esperado visto que o PSP está sendo apoiado por uma ferramenta de 2ª geração.

5.3. Descrição do Estudo de Caso

O Estudo de Caso teve a exata duração de 30 dias, período no qual foi acompanhado um projeto desenvolvido em Java por apenas um programador. Para a coleta das métricas de tempo, foram utilizadas duas ferramentas simultaneamente: a ferramenta de 2ª Geração *Process Dashboard* e a ferramenta de 3ª Geração *Hackystat*. A *Dashboard* já era habitualmente utilizada pelo programador em suas atividades cotidianas de desenvolvimento de software há dois anos, enquanto a *Hackystat* foi instalada e configurada em seu ambiente de desenvolvimento pela primeira vez. O uso simultâneo das duas ferramentas não sobrecarregou o programador, visto que a

ferramenta de 3ª geração *Hackystat*, uma vez instalada, coletava os dados automaticamente e sem a intervenção do programador.

Para a análise dos dados, o programador enviou ao pesquisador (autor deste trabalho) um arquivo contendo os dados coletados localmente pela ferramenta *Dashboard* e também o nome de usuário e senha pessoais do programador utilizados para o acesso ao servidor público da *Hackystat*. Desta maneira, o pesquisador consultava os dados coletados pela *Hackystat* diretamente do servidor.

O foco da análise dos dados neste estudo foi a fase de codificação, uma vez que a ferramenta *Hackystat* não coleta o tempo das fases de Planejamento e Análise Final. A ferramenta *Process Dashboard*, por outro lado, é capaz de registrar o tempo de todas as fases, pois quem determina o início e o término de cada fase é o próprio desenvolvedor.

Ao final do período de 30 dias, foram comparadas as medidas registradas por ambas as ferramentas, na tentativa de se identificar tendências ou relações entre tais medidas. No entanto, reforça-se a necessidade de elaborar mais Estudos de Caso em diferentes ambientes e com um número maior de participantes para que as análises aqui realizadas sejam generalizadas.

5.4. Análise de Resultados

Apesar de o foco ser a codificação, na Tabela 25 apresentam-se os valores, em minutos, coletados para todas as fases.

Tabela 25 - Métricas de tempo registradas pelas ferramentas Dashboard e *Hackystat* (em minutos)

Descrição	Dashboard	Hackystat	Diferença absoluta	Diferença percentual
Planejamento	154	-	-	-
Projeto	57	-	-	-
Codificação	1108	1140	32	2,88%
Testes	264	-	-	-
Análise Final	70	-	-	-
Total	**1743**	**1140**	**603**	**-34,59%**

Se observado o tempo total registrado, na *Dashboard* esse valor é bem maior porque esta ferramenta é capaz de registrar o tempo de todas as fases. Como mostra a Tabela 25, foram registrados 603 minutos (34,50%) totais a mais na *Dashboard*. No entanto, quando se compara apenas a fase de codificação de ambas as ferramentas, a

diferença é bem menor, sendo que a *Hackystat* registrou somente 32 minutos (2,88%) a mais do que a *Dashboard*.

Também foi feito um levantamento de coleta de tempo por dia. Apresentam-se, na Tabela 26, os resultados dos dias mais e menos ativos de acordo com as duas ferramentas.

Tabela 26 - Tempo registrado pelas ferramentas Dashboard e Hackystat nos dias mais e menos ativos (em minutos)

Descrição	Dashboard	Hackystat	Diferença absoluta	Diferença percentual
Dia mais ativo	293	240	-53	-18,08%
2º Dia mais ativo	248	150	-98	-39,51%
2º Dia menos ativo	40	30	-10	-25%
Dia menos ativo	2	144	142	7100%

Como pode ser observado, em geral a ferramenta *Hackystat* captura menos tempo do que o programador registra na *Dashboard*. Uma possível explicação pode ser o fato de que a *Hackystat* é preparada para capturar separadamente o tempo de vários projetos que são desenvolvidos simultaneamente, além de ser capaz de detectar automaticamente pequenas interrupções no desenvolvimento. Já a ferramenta *Dashboard* marca o tempo continuamente, inclusive quando o desenvolvedor trabalha por um pequeno tempo em outros projetos ou está falando ao telefone, a menos que este sempre se lembre de interromper o cronômetro quando realiza outra atividade. Como essas atividades paralelas são, geralmente, de curta duração, é improvável que o desenvolvedor se lembre ou tenha o cuidado de gerenciar o cronômetro corretamente em todas as ocasiões.

No entanto, há uma grande disparidade no dia menos ativo, pois a *Dashboard* registrou apenas 2 minutos de desenvolvimento, enquanto a *Hackystat* registrou 144, o que corresponde a um erro de 7.100%. Uma investigação mais precisa nos dados registrados pelo desenvolvedor nesse dia mostra que mais de 6 horas de desenvolvimento foram registradas em outro projeto na *Dashboard*. Isso é um forte indicativo de que o cronômetro da *Dashboard* fora ligado em um projeto diferente do projeto em que realmente o desenvolvimento estava acontecendo. Na *Hackystat*, fica claro que os arquivos de código-fonte editados pertencem, efetivamente, ao projeto que faz parte deste Estudo de Caso. O programador que participou do Estudo de Caso afirmou que medidas atípicas como esta são registradas na *Dashboard* geralmente

quando ocorre algum problema no curso normal de desenvolvimento, como por exemplo um módulo que apresenta defeito e que precisa ser corrigido urgentemente, ou quando é necessário desenvolver software usando uma arquitetura ou tecnologia cujo conhecimento não é completamente dominado.

No período de 30 dias que constitui este estudo de caso, foi registrado pelo desenvolvedor na ferramenta *Dashboard* que 58% do total de seu tempo foram dedicados à atividade de suporte a usuários, tempo este que não tem relação com o desenvolvimento de nenhum projeto específico. Esse é outro tipo de informação que a ferramenta *Hackystat* não coleta e que é importante para compor os dados históricos da empresa.

Nesse mesmo período, se os dados forem analisados fora do contexto de um projeto específico, incluindo todas as atividades realizadas pelo desenvolvedor e até mesmo as atividades de suporte ao cliente, o total de tempo registrado foi de 3555 minutos na *Dashboard*. Já na *Hackystat*, no mesmo período, foi de 3180 minutos. O erro, neste caso, é de apenas -10,54%, menor do que os -34,59% se analisarmos somente o projeto considerado no estudo de caso. É provável que a incidência de atividades de codificação tenha sido predominante nesse período e, apesar da *Hackystat* não possibilitar o registro de tempo de todas as fases, são necessários mais estudos para determinar a causa desse menor erro quando o tempo é analisado fora do contexto de um projeto específico.

5.5. Identificação dos benefícios e dificuldades

A primeira dificuldade encontrada foi que um dos programadores da empresa não pôde participar do Estudo de Caso por problemas técnicos em seu ambiente de desenvolvimento. A ferramenta *Hackystat* foi instalada com sucesso em seu computador, no entanto foi impossível fazer com que funcionasse a comunicação entre sensor e servidor. Sem essa comunicação, os dados ficavam sendo armazenados localmente, ocupando muito espaço em disco (cerca de 10GB). Como os dados não estavam sendo enviados para o servidor, a *Hackystat* não analisava os dados. Várias soluções técnicas envolvendo configurações de *firewall* e do sistema operacional da máquina do programador foram testadas, mas foi impossível resolver o problema e, portanto, não foi possível que o programador participasse do Estudo de Caso.

Outro problema identificado é que há um grande número de atividades do PSP que não têm suporte da ferramenta *Hackystat*, como as atividades realizadas na fase de Planejamento. Dessa maneira, a fase de Análise Final também fica bastante comprometida, uma vez que nessa fase, basicamente, se compara o desenvolvimento efetivamente realizado com as estimativas realizadas na fase de Planejamento. Outras atividades do PSP que não são suportadas pela *Hackystat* são o PIP, o método de estimativa PROBE, a elaboração de cronogramas e o fornecimento e customização de *checklists* de Revisão. A Revisão de Código é suportada pela *Hackystat*, mas tal atividade não foi utilizada no Estudo de Caso pois os programadores desenvolvem atividades do Nível 1.1 do PSP, e Revisão de Código pertence ao Nível 2.

O benefício identificado durante esse Estudo de Caso é que foi possível utilizar simultaneamente tanto uma ferramenta de 2ª geração quanto uma ferramenta de 3ª geração, sem que houvesse sobrecarga para o desenvolvedor. Dessa maneira, abre-se a possibilidade de mesclar ambos os tipos de abordagem em uma só ferramenta, com o intuito de dar suporte a todas as atividades do PSP, ao mesmo tempo em que se automatizam as atividades possíveis. De fato, é interessante que se automatizem o máximo de atividades da fase de Codificação, que é a fase em que os programadores mais reclamam de interferência do PSP. Por outro lado, as fases de Planejamento e Análise Final podem conter um número maior de atividades não automatizadas.

A *Hackystat*, por ser uma ferramenta código-aberto, poderia ser facilmente adaptada para apresentar características de ferramentas de 2ª geração. Essa abordagem 'mista' que combina dados coletados manualmente com dados coletados automaticamente pode melhorar alguns aspectos da medição, como discutido por Hochstein et. al [Hochstein, 2005].

5.6. Considerações Finais

Apresentou-se neste capítulo o Estudo de Caso sobre o uso do PSP e ferramentas de apoio em uma pequena empresa de desenvolvimento de software. Como o Estudo de Caso foi realizado com somente um programador, não é viável generalizar os dados coletados e analisados para qualquer grupo de programadores. Assim, o foco foi alterado para que fosse realizado um comparativo entre medidas coletadas por ferramentas de apoio ao PSP de diferentes gerações.

Os resultados mostraram que a diferença entre as métricas não foram tão significativas quanto se esperava. Em um projeto de cerca de 20 horas, a diferença de medida de tempo registrada na fase de codificação entre as duas ferramentas foi de apenas 32 minutos, ou 2,88%. Mas a análise específica de curtos períodos de tempo (no caso do Estudo de Caso, a análise do tempo diário) mostra que a diferença é sim significativa, mas que os erros variam ora para mais, ora para menos, fazendo com que uma análise em longo prazo tenha a tendência de aproximar os valores.

Além da diferença das medidas coletadas por ferramentas de diferentes gerações, constatou-se que a ferramenta *Hackystat* dá suporte a um número muito pequeno de atividades do PSP, e que uma reaproximação entre ferramentas ISEMA e o PSP não só é necessária, como é possível através de uma série de modificações na ferramenta e no ambiente de trabalho do desenvolvedor.

CAPÍTULO 6

CONCLUSÕES

Este trabalho teve por objetivo avaliar o impacto do uso de ferramentas de 3ª geração na aplicação do PSP, tanto em meio acadêmico quanto em empresas de desenvolvimento de software. As ferramentas de 3ª geração caracterizam-se por coletar e analisar métricas de Engenharia de Software automaticamente e sem obstruir o trabalho do desenvolvedor. Essa característica de "invisibilidade" da ferramenta é muito importante durante a aplicação do PSP, pois evita que o desenvolvedor tenha que constantemente interromper o seu trabalho para coletar e analisar métricas. No entanto, nem todas as atividades do PSP podem ser automatizadas. Assim, tais ferramentas dão suporte a um número limitado de atividades do PSP e, por este motivo, acabaram se distanciando do objetivo de dar apoio ao PSP e se tornaram ferramentas ISEMA (*In-process Software Engineering Measurement and Analysys*).

Assim, para atingir o objetivo, este trabalho realizou dois Estudos de Caso - um em meio acadêmico e outro em contexto de uma empresa de desenvolvimento de software - uma vez que Estudos de Casos podem identificar várias tendências ou características de um método, ferramenta, abordagem ou processo. Para planejar os Estudos de Caso de maneira sistemática e determinar quais características deveriam ser identificadas e medidas, foi utilizado o paradigma GQM. Assim, procurou-se garantir que os recursos disponíveis seriam utilizados na coleta e análise de dados relevantes e úteis para este trabalho. O planejamento GQM foi apresentado no Capítulo 3.

Os resultados apresentados pelos Estudos de Caso foram uma mistura de resultados esperados com novas descobertas. Os resultados esperados muitas vezes corroboraram com os resultados de trabalhos relacionados, dos quais muitos são também Estudos de Caso sobre PSP. No entanto, como ferramentas ISEMA apresentam um conceito relativamente novo na Engenharia de Software, foi possível identificar resultados significativos que vêm acrescentar informações relevantes às áreas abordadas

neste trabalho. Em especial, não se conhece nenhuma tentativa anterior a este trabalho em readaptar ferramentas ISEMA para o PSP.

O Estudo de Caso 1 foi realizado em ambiente acadêmico em duas turmas de disciplinas em que se realizaram atividades de desenvolvimento de software. O principal objetivo desse Estudo de Caso foi identificar, na prática, as dificuldades que existem quando se utilizam ferramentas de 3ª geração e PSP. Verificou-se que, uma vez instaladas, as ferramentas de 3ª geração realmente geram pouca ou nenhuma sobrecarga no trabalho do desenvolvedor. No entanto, a própria instalação e configuração da ferramenta se mostraram uma sobrecarga de atividades para os alunos. Esse Estudo de Caso foi apresentado no Capítulo 4.

O Estudo de Caso 2 foi realizado em uma pequena empresa de desenvolvimento de software da cidade de São Carlos. Como essa empresa já utilizava o PSP com ferramentas de 2ª geração, a ferramenta *Hackystat* foi utilizada em paralelo com a ferramenta *Process Dashboard*. Desta maneira, foi possível coletar dados de um mesmo projeto a partir das duas ferramentas. No caso, compararam-se medidas de tempo, e embora a diferença registrada entre as duas ferramentas tenha sido de apenas 2,88% na fase de codificação, constatou-se que mesmo uma pessoa muito motivada para usar o PSP e comprometida em coletar os dados da maneira mais precisa possível comete erros ao usar ferramentas de 2ª geração. Também foi possível identificar as atividades do PSP que não são apoiadas pela ferramenta *Hackystat* (e outras ferramentas ISEMA em geral) e propõe-se, portanto, uma fusão de conceitos de ferramentas de 2ª e 3ª geração em uma única ferramenta de apoio.

Assim, o trabalho aqui apresentado foi um primeiro passo para se fazer um levantamento das dificuldades que deverão ser tratadas antecipadamente para que se consiga implantar o uso do PSP na grade curricular dos cursos de computação desta universidade, já que o objetivo maior é esse. Da mesma forma, no contexto de pequenas empresas, considera-se que o uso do PSP seja uma iniciativa significante para facilitar a implantação de um plano de melhoria de processo. No caso da empresa que participou do Estudo de Caso aqui apresentado, o fato de usar o PSP já permitiu que o processo de planejamento se tornasse um processo bem definido, que vem sendo utilizado por ela.

6.1. Contribuições do Trabalho

A seguir relacionam-se, resumidamente, algumas contribuições deste trabalho:

- Proposta de reengenharia de ferramentas ISEMA para que estas dêem suporte ao PSP;
- Identificação das maiores dificuldades dos alunos na aplicação do PSP;
- Identificação da importância em dar suporte adequado aos alunos para a instalação das ferramentas de apoio;
- Identificação de impacto negativo em relação à instalação das ferramentas de apoio acima do esperado;
- Identificação das atividades do PSP que não são apoiadas pela ferramenta *Hackystat* (e outras ferramentas ISEMA em geral);
- Comparação entre métricas coletadas por diferentes gerações de ferramentas mostram que há grande diferença em períodos pequenos, mas que os erros para mais e para menos tendem a se anular em longo prazo;

6.2. Trabalhos Futuros

A seguir relacionam-se aspectos que fornecem perspectivas de continuidade deste trabalho:

- Adaptar a ferramenta *Hackystat* para que esta implemente as fases de Planejamento e Análise Final do PSP;
- Aumentar a integração entre o ambiente de desenvolvimento, o sensor e o servidor *Hackystat* para diminuir a quantidade de visitas necessárias à página Web do servidor para a configuração de projetos e consulta de análises;
- Elaborar sensores compatíveis com a ferramenta *Hackystat* para outros softwares tais como clientes de email (usado no suporte à cliente), navegadores de Internet (usado em pesquisas) e IDEs (prioridade para DevC++ e Netbeans);
- Elaboração de mais Estudos de Caso em meio acadêmico, acompanhando o uso do PSP dos alunos por um período maior do que um semestre;
- Elaboração de mais Estudos de Caso em empresas de software com equipes maiores e por um período de tempo mais longo;
- Desenvolver uma estratégia de implantação do PSP apoiado por ferramentas de 3ª geração no currículo de cursos de graduação em computação;

REFERÊNCIAS BIBLIOGRÁFICAS

[Babar & Potter, 2005] Babar, A. & Potter, J. Adapting the personal software process (PSP) to formal methods, Software Engineering Conference, 2005. Proceedings. 2005 Australian, 2005, 192-201

[Basili, 1992] Basili, V. R. 1992 *Software Modeling and Measurement: the Goal/Question/Metric Paradigm*. Technical Report. University of Maryland at College Park.

[Basili, 1994] V. Basili, GQM approach has evolved to include models, IEEE Software, vol. 11, no. 1, pp. 8, 1994.

[Basili, Caldiera, Rombach, 1994] V. Basili, G. Caldiera, and H.D. Rombach, "Goal Question Metric Approach," Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc., 1994

[Basili et. al., 2007] Basili, V., Heidrich, J., Lindvall, M., Munch, J., Regardie, M., and Trendowicz, A. 2007. GQM^+ Strategies -- Aligning Business Strategies with Software Measurement. In Proceedings of the First international Symposium on Empirical Software Engineering and Measurement (September 20 - 21, 2007). ESEM. IEEE Computer Society, Washington, DC, 488-490.

[Börstler et. al., 2002] Borstler, J.; Carrington, D.; Hislop, G.; Lisack, S.; Olson, K. & Williams, L. Teaching PSP: challenges and lessons learned, Software, IEEE, 2002, 19, 42-48

[Cannon, 1999] Cannon, R. Putting the Personal Software Process into practice, Software Engineering Education and Training, 1999. Proceedings. 12th Conference on, 1999, 34-37

[Carrington et. al., 2001] Carrington, D.; McEniery, B. & Johnston, D. PSP in the large class, Software Engineering Education and Training, 2001. Proceedings. 14th Conference on, 2001, 81-88

[Ceberio-Verghese, 1996] Ceberio-Verghese, A. Personal Software Process: a user's perspective Software Engineering Education, 1996. Proceedings., Ninth Conference on, 1996, 52-65

[Chrissis et. al., 2004] Ben-Menachem, M. 2004. Review of "CMMI: guidelines for process integration and product improvement by Mary Beth Chrissis, Mike Konrad and Sandy Shrum." Addison Wesley 2003. SIGSOFT Softw. Eng. Notes 29, 5 (Sep. 2004), 37-38. DOI= http://doi.acm.org/10.1145/1022494.1022552

[Davila-Nicanor & Mejia-Alvarez, 2004] Davila-Nicanor, L. & Mejia-Alvarez, P. Reliability improvement of Web-based software applications Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on, 2004, 180-188

[Eman et. al., 1996] El Emam, K.; Shostak, B. & Madhavji, N. Implementing concepts from the Personal Software Process in an industrial setting, Software Process, 1996. Proceedings., Fourth International Conference on the, 1996, 117-130

[Escala & Morisio, 1999] Escala, D. & Morisio, M. A metric suite for a team PSP, Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International, 1998, 89-92

[Fergunson et. al., 1997] Ferguson, P.; Humphrey, W.; Khajenoori, S.; Macke, S. & Matvya, A. Results of applying the Personal Software Process, Computer, 1997, 30, 24-31

[Hayes, 1998] Hayes, W. Using a Personal Software Process to improve performance, Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International, 1998, 61-71

[Hilburn, 1999] Hilburn, T. PSP metrics in support of software engineering education, Software Engineering Education and Training, 1999. Proceedings. 12th Conference on, 1999, 135-136

[Hochstein, 2005] Hochstein, L., Basili, V. R., Zelkowitz, M. V., Hollingsworth, J. K., and Carver, J. 2005. Combining self-reported and automatic data to improve programming effort measurement. In Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT international Symposium on Foundations of Software Engineering (Lisbon, Portugal, September 05 - 09, 2005). ESEC/FSE-13. ACM, New York, NY, 356-365.

[Humphrey, 1994] Humphrey, W. The personal process in software engineering, Software Process, 1994, 'Applying the Software Process', Proceedings., Third International Conference on the, 1994, 69-77

[Humphrey, 1995] Humphrey, W. Making process improvement personal, Software, IEEE, 1995, 12, 82-83

[Humphrey, 1995b] Humphrey, W., A Discipline for Software Engineering, Addison-Wesley Professional, 1st edition, (December, 1995)

[Humphrey, 1997] Humphrey, W., Introduction to the Personal Software Process, Addison-Wesley Professional, 1st edition, (December, 1996)

[Johnson & Disney, 1998] Johnson, P. & Disney, A. The personal software process: a cautionary case study, Software, IEEE, 1998, 15, 85-88

[Johnson, 1999] Johnson, P. M. 1999. Leap: a "personal information environment" for software engineers. In Proceedings of the 21st international Conference on Software Engineering (Los Angeles, California, United States, May 16 - 22, 1999). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 654-657.

[Johnson et. al., 2003] Johnson, P.; Kou, H.; Agustin, J.; Chan, C.; Moore, C.; Miglani, J.; Zhen, S. & Doane, W. Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined, Software Engineering, 2003. Proceedings. 25th International Conference on, 2003, 641-646

[Lisack, 1999] Lisack, S. The Personal Software Process in the classroom: student reactions (an experience report), Software Engineering Education & Training, 2000. Proceedings. 13th Conference on, 2000, 169-175

[Maletic et. al., 2001] Maletic, J.; Howald, A. & Marcus, A. Incorporating PSP into a traditional software engineering course: an experience report, Software Engineering Education and Training, 2001. Proceedings. 14th Conference on, 2001, 89-97

[Mengel, 1999] Mengel, S. Software metrics: view from education, research and training Software Engineering Education and Training, 1999. Proceedings. 12th Conference on, 1999, 126-128

[Morisio, 2000] Morisio, M. 2000. Applying the PSP in Industry. IEEE Softw. 17, 6 (Nov. 2000), 90-95.

[MOODLE, 2008] http://moodle.org/, último acesso em 21/02/2008.

[Paulk et. al., 1993] Paulk, M. C., Weber, C. V., Garcia, S. M., Chrissis, M. B., Bush, M. "Key Practices of the Capability Maturity Model" Version 1.1". Research Access for Software Engineering Institute, 1993.

[Prechelt & Unger, 2000] Prechelt, L.; Unger, B.; An experiment measuring the effects of personal software process (PSP) training, IEEE Transactions on Software Engineering, 2001, Volume 27, Issue 5, 465-472

[Process Dashboard, 2008] Ferramenta Process Dashboard – 2008 – Disponível em http://www.processdash.com/, último acesso em 04/03/2008.

[Runeson, 2001] Runeson, P. Experiences from teaching PSP for freshmen, Software Engineering Education and Training, 2001. Proceedings. 14th Conference on, 2001, 98-107

[Sanchez et. al., 2007] Sanchez, A.; Montebelo, R.; Fabbri, S. PCU|PSP: Uma Estratégia para ajustar Pontos por Casos de Uso por meio do PSP em Empresas de Pequeno Porte, Simpósio Brasileiro de Qualidade de Software, 2007, Porto de Galinhas, Brasil. Anais, p. 187-202.

[Sun, 2007] Java Technology – 2007 – Disponível em:

http://java.sun.com/

Último acesso em 28/02/2007

[Sun, 2007b] JavaServer Faces Technology – 2007 – Disponível em:

http://java.sun.com/javaee/javaserverfaces/

Último acesso em 28/02/2007

[Sun, 2007c] http://jcp.org/en/jsr/detail?id=220, último acesso em 22/02/2008

[Tomayko, 2003] Tomayko, J. Scientific management meets the personal software process, Software, IEEE, 2003, 20, 12-14

[Towhidnejad & Hilburn, 1997] Towhidnejad, M. & Hilburn, T. Integrating the Personal Software Process (PSP) across the undergraduate curriculum, Frontiers in Education Conference, 1997. 27th Annual Conference. 'Teaching and Learning in an Era of Change'. Proceedings., 1997, 1, 162-168vol.1

[W3, 2007] Simple Object Access Protocol – 2003 – Disponível em:

http://www.w3.org/TR/soap/

Último acesso em 28/02/2007

[Williams, 1997] Williams, L. A. 1997. Adjusting the instruction of the personal software process to improve student participation. In Proceedings of the Frontiers in Education Conference,1997. on 27th Annual Conference. Teaching and Learning in An Era of Change. - Volume 01 (November 05 - 08, 1997). FIE. IEEE Computer Society, Washington, DC, 154-156vol.1.

[Wohlin et. al., 2000] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. 2000 Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers.

[Zhong et. al., 2000] Zhong, X.; Madhavji, N. & El Emam, K. Critical factors affecting personal software processes Software, IEEE, 2000, 17, 76-83

APÊNDICE A

Neste apêndice apresenta-se o questionário aplicado nos participantes do Estudo de Caso.

Questionário do Experimento de PSP e da Ferramenta *Hackystat*

Cada uma das seguintes questões tem como objetivo caracterizar a população participante do experimento. Por favor, responda a cada uma das questões honestamente. O resultado do questionário será mantido em anonimato e nenhuma análise individual será realizada. Escolha somente uma opção para cada questão, exceto quando se destaca o contrário.

Instituição:_____Disciplina:_____
Professor:_____Turma:_____

1. Qual sua experiência como programador no *meio acadêmico*?

[ ] – Sem experiência (0-7 programas)
[ ] – Pouca experiência (8-15 programas)
[ ] – Experiência intermediária(16-20 programas)
[ ] – Muita experiência (21 ou + programas)
[ ] – Não sei

2. Qual sua experiência como programador na *indústria de software*?

[ ] – Sem experiência
[ ] – Pouca experiência (até 1 ano)
[ ] – Experiência intermediária (1 a 2 anos)
[ ] – Muita experiência (2 anos ou mais)

3. Em qual perfil de seu curso de graduação você se encaixa?

[ ] – 1º ano de graduação
[ ] – 2º ano de graduação
[ ] – 3º ano de graduação
[ ] – 4º ou 5º ano de graduação
[ ] – Formado
[ ] – Pós-Graduação
[ ] – Nenhuma das alternativas (complete): _____

4. Qual seu curso de graduação?

[ ] – Ciência da Computação
[ ] – Engenharia de Computação
[ ] – Informática

[ ] – Sistemas de Informação
[ ] – Outro:_____

5. Quais dos seguintes ambientes de programação você costuma utilizar? **(marque todos que se apliquem)**

[ ] – Eclipse
[ ] – NetBeans
[ ] – JDeveloper
[ ] - JBuilder
[ ] – Crimson Editor
[ ] – DevC++ / Dev-Pascal
[ ] – Vi / Vim
[ ] – Emacs
[ ] – Visual Studio
[ ] – Borland C++
[ ] – Turbo C / Turbo C++
[ ] – Turbo Pascal
[ ] – Delphi
[ ] – Dreamweaver
[ ] – Frontpage
[ ] - Coldfusion
[ ] – Outros:_____

6. Como você julga sua capacidade de estimar prazos e/ou custos?

[ ] – Não estimo prazos e/ou custos
[ ] – Dificilmente acerto prazos e/ou custos previstos
[ ] – Tenho algum sucesso prevendo prazos e/ou custos
[ ] – Geralmente tenho sucesso prevendo prazos e/ou custos

7. Caso você estime prazos e/ou custos, qual a sua principal maneira de realizar essa estimativa?

[ ] – Intuição ou experiência
[ ] – Uso dados históricos pessoais ou de terceiros
[ ] – Uso um processo bem definido

8. Como você se preocupa com a qualidade de seu trabalho?

[ ] – Não me preocupo com a qualidade
[ ] – Tento fazer o meu melhor a todo instante, mas não tento melhorar formalmente através de uma técnica ou processo
[ ] – Estou tentando melhorar formalmente através de uma técnica ou processo

9. Você considera métodos de melhoria contínua de desenvolvimento de software (como CMMI, PSP) **importantes**?

[ ] – Sim [ ] – Não [ ] – Não sei

10. Você considera métodos de melhoria contínua de desenvolvimento de software (como CMMI, PSP) **interessantes**?

[ ] – Sim [ ] – Não [ ] – Não sei

11. Quanto você *conhece* o método PSP?

[ ] – Não conheço
[ ] – Conheço pouco

[ ] – Conheço razoavelmente bem
[ ] – Conheço completamente ou participei de um treinamento

12. Qual sua opinião geral sobre o PSP?

[ ] – Nunca usei o PSP e não tenho interesse em usá-lo
[ ] – Nunca usei o PSP, mas tenho interesse em usá-lo
[ ] – Gostei de minha experiência com o PSP, mas não pretendo voltar a usá-lo espontaneamente
[ ] – Gostei de minha experiência com o PSP e pretendo voltar a usá-lo espontaneamente
[ ] – Não gostei de minha experiência com o PSP e não pretendo voltar a usá-lo espontaneamente
[ ] – Não gostei de minha experiência com o PSP, mas pretendo voltar a usá-lo espontaneamente
[ ] – Não tenho opinião sobre o PSP

Caso nunca tenha usado o PSP, pule para a questão 18. As questões 13-17 são para quem está atualmente usando o PSP ou para quem já usou o PSP no passado.

13. Como você *usou* o método PSP? **(marcar todas que se apliquem)**

[ ] – Usei espontaneamente no meio acadêmico
[ ] – Usei espontaneamente na indústria
[ ] – Usei espontaneamente em projetos pessoais
[ ] – Fui forçado a usar no meio acadêmico
[ ] – Fui forçado a usar na indústria

14. Como você avalia a ferramenta de apoio usada durante o uso do PSP?

[ ] – A ferramenta não ajudou durante a aplicação do PSP
[ ] – A ferramenta pouco ajudou durante a aplicação do PSP
[ ] – A ferramenta ajudou consideravelmente durante a aplicação do PSP
[ ] – A ferramenta atrapalhou a aplicação do PSP
[ ] – Não sei responder

15. Em qual nível do PSP você se encaixa?

[ ] – PSP nível 0 ou 0.1
[ ] – PSP nível 1 ou 1.1
[ ] – PSP nível 2 ou 2.1
[ ] - PSP nível 3
[ ] – Usei o PSP, mas não me encaixo em nenhum nível específico (realizo atividades de diversos níveis diferentes)
[ ] – Não sei

16. Como o PSP influenciou o seu trabalho? (**marcar todas que se apliquem)**

[ ] – Não houve influência
[ ] – O PSP me ajudou a planejar e estimar melhor os meus projetos

[ ] – O PSP me ajudou na qualidade do meu trabalho, ou seja, cometo menos erros do que antes
[ ] – O PSP me ajudou a ser mais produtivo, ou seja, consigo terminar as tarefas em um tempo menor do que antes
[ ] – O PSP me influenciou negativamente em algum aspecto

17. Avaliando as maiores dificuldades para se usar o PSP, classifique cada um dos seguintes itens de "1" (não foi um problema) até "5" (muito problemático):

Ferramenta de apoio....._____
Conceitos do PSP....._____
Motivação....._____
Interferência no trabalho....._____
Dificuldade em compreender como o PSP pode me ajudar....._____

Caso você tenha usado o PSP, pule para a questão 19. A questão 18 é destinada para quem nunca usou o PSP.

18. Qual o principal motivo para a não utilização do PSP na disciplina?

[ ] – Falta de interesse no PSP
[ ] – Falta de tempo
[ ] – Problemas com as ferramentas de apoio
[ ] – Outro:_____

19. O espaço a seguir é reservado para qualquer comentário, crítica ou dúvida sobre o PSP, ferramentas de apoio, sobre o experimento e sobre este questionário.

Obrigado pelas suas respostas!!

APÊNDICE B

Neste apêndice apresentam-se todos os materiais desenvolvidos para auxiliar os alunos no Estudo de Caso.

Instalando e Configurando o Eclipse para uso com a linguagem C++ no ambiente Windows

O objetivo desse documento é detalhar os procedimentos necessários para a correta instalação e configuração do Eclipse para uso com a linguagem C++. O Eclipse (www.eclipse.org) é apenas uma IDE de programação, não um compilador. Portanto, será necessário instalar também um compilador C++. Neste tutorial, vamos utilizar o compilador Mingw, que é uma versão do GNU gcc para Windows. É o mesmo compilador usado pelo "Dev-C++".

Instalando o compilador (Dev-C++)

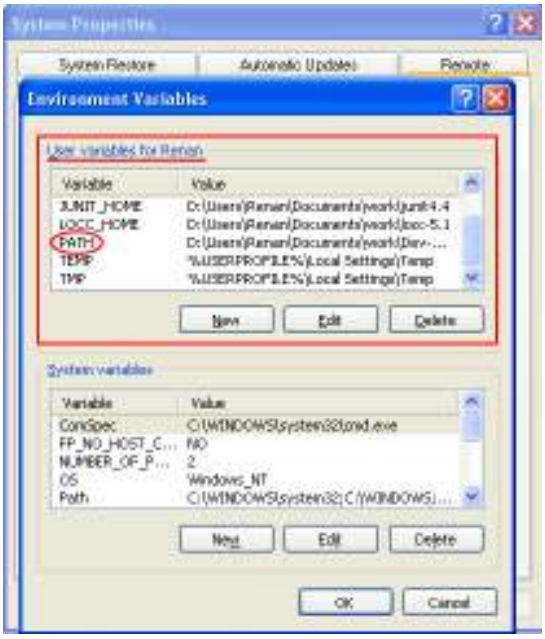
Usuários do LIG do DC não precisam realizar esta etapa, pois o Dev-C++ já se encontra instalado em "C:\Dev-Cpp."

A maneira mais fácil de instalar o Mingw é instalando o próprio Dev-C++. Uma cópia da última versão do Dev-C++ está disponível em http://www.dc.ufscar.br/~renan_montebelo/psp/. Alternativamente, o site oficial de download do Dev-C++ é http://www.bloodshed.net/dev/devcpp.html. Basta realizar o download do arquivo apropriado, executá-lo e seguir as instruções. ATENÇÃO: não instale o Dev-C++ em um caminho de diretório que contenha espaços em branco, como em "C:\Documentos and Settings\renan\devcpp". O diretório padrão de instalação é "C:\Dev-Cpp".

Configurando o PATH do compilador

Esta etapa deve ser executada, inclusive no Lig, após a instalação do Dev-C++. Este passo é essencial para que o Eclipse "encontre" a instalação do compilador.

1. Clicar com o botão direito sobre o ícone do “Meu Computador” (no desktop) e depois clicar em “Propriedades”. Alternativamente, pressione simultaneamente, no teclado, as teclas “Windows” e “Pause/Break”.
2. Selecione a *tab* “Avançado”.
3. Pressione o botão “Variáveis de Ambiente”.
4. A tela apresentada é dividida em duas seções: variáveis do usuário e variáveis do sistema. Procure na seção de variáveis do usuário pela variável de sistema “PATH”.



   a. Caso PATH já exista, selecione-a e clique em editar. Ao final do valor da variável, adicione um ponto-e-vírgula e logo depois insira o diretório de instalação do Dev-C++ acrescido do diretório bin. Exemplo: “xxx;yyy;zzz;C:\Programas\Dev-Cpp\bin”. Não coloque aspas e não coloque espaço antes ou depois do ponto-e-vírgula inserido. X, Y e Z exemplificam quaisquer seqüências de caracteres que, eventualmente, já estavam configuradas na variável PATH;
   b. Caso PATH não exista, clique em “Novo”, coloque o nome da nova variável como PATH e o valor da mesma sendo o diretório de instalação do Dev-C++ acrescido do diretório bin. Exemplo: “C:\Programas\Dev-Cpp\bin”. Não coloque aspas.
5. Clique em OK em todas as janelas.

Instalando o Java Runtime Environment

Essa etapa é desnecessária no Lig, uma vez que o Java já se encontra instalado.

O Eclipse necessita do Java instalado para funcionar. Para instalar o Java, basta acessar a página www.java.com e seguir as instruções. Qualquer JRE versão 5 (1.5.0) ou maior é suficiente. No entanto, é recomendável ter sempre a última versão instalada. Atualmente a última versão é a 6 (1.6.0_02).

Instalando o Eclipse

Existem duas opções para instalar o Eclipse.

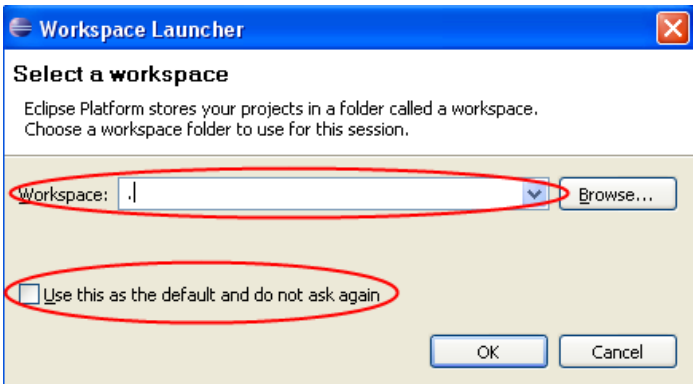
Opção 1 (Recomendável):

1. Acessar http://www.dc.ufscar.br/~renan_montebelo/psp/. Fazer o download do Eclipse.
2. Descompactar o arquivo eclipse_dc.zip em um diretório de sua escolha. ATENÇÃO: usuários do Lig devem instalar o Eclipse em "C:\Documents and Settings*LoginDoUsuárioDoLig*\eclipse". Isso torna possível utilizar o Eclipse em qualquer máquina do Lig. Uma vez feito o login em qualquer máquina, a instalação do Eclipse "acompanhará" o perfil do usuário, com todas as suas configurações.
3. Uma vez descompactado, o Eclipse está pronto para uso. Esta versão do Eclipse (eclipse_dc.zip) já contém as configurações necessárias para uso do Eclipse com Java e C++.

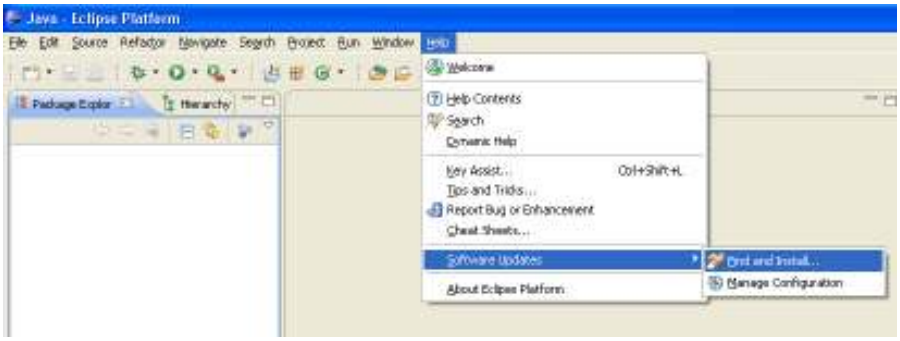
Opção 2: Download direto do site do Eclipse (não funciona no Lig):

1. Acessar www.eclipse.org/. Na seção de downloads, efetuar o download do **"Eclipse IDE for Java Developers"**. NÃO faça o download do "Eclipse IDE for C/C++ Developers", pois não vai funcionar no contexto do experimento PSP proposto.
2. Descompactar o arquivo .zip em um diretório de sua escolha.

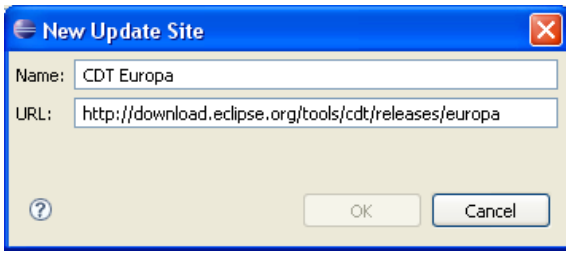
3. Executar o Eclipse (se perguntado sobre o Workspace, utilize apenas um "ponto final" por ora, e não ative o checkbox, como na figura abaixo).



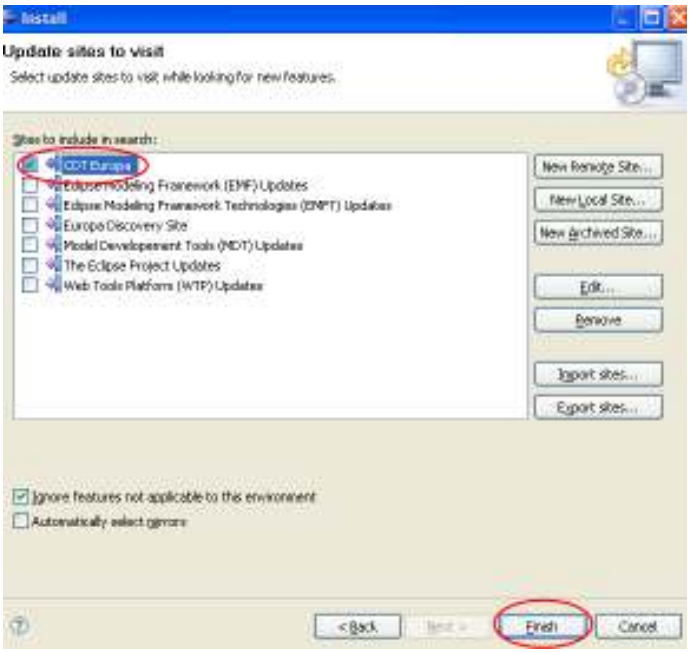
4. Com o Eclipse aberto, acesse Help → Software Updates → Find and Install



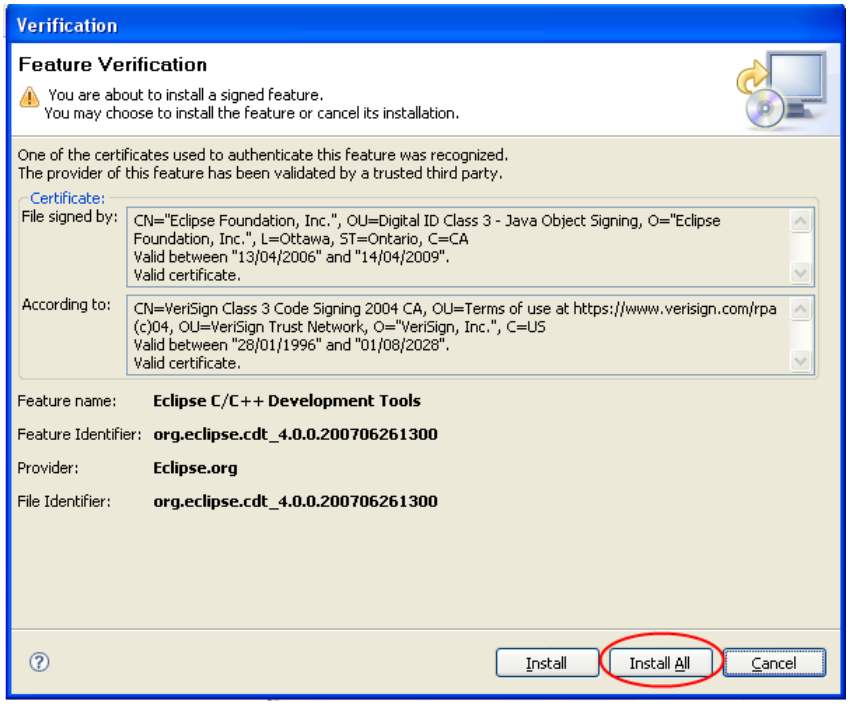
5. Selecione "Search for new features to install" e clique em Next.
6. Clique em "New Remote Site". Coloque como nome "CDT Europa" e como endereço http://download.eclipse.org/tools/cdt/releases/europa. Clique em OK.



7. Selecione somente "CDT Europa" e clique em "Finish".



8. Aguarde o Eclipse terminar a checagem do site. Aparecerá uma nova tela, selecione "CDT Europa" novamente. Aguarde nova checagem do Eclipse. Clique em Next.
9. Aceite os termos e clique em Next.
10. Em uma nova tela, clique em "Finish".
11. O Eclipse fará o download do pacote CDT.
12. Uma tela de verificação de pacote aparecerá. Clique em "Install All".



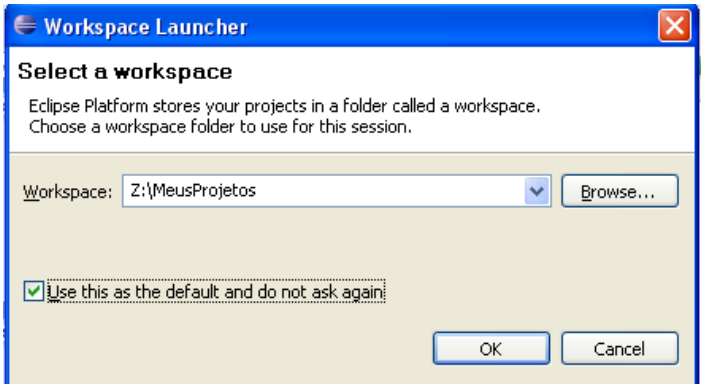
13. O Eclipse vai instalar os pacotes. Logo após, o Eclipse vai perguntar se deseja reiniciar o Eclipse. Clique em "Yes".

14. Após o reinicio do Eclipse, o mesmo está pronto para uso.

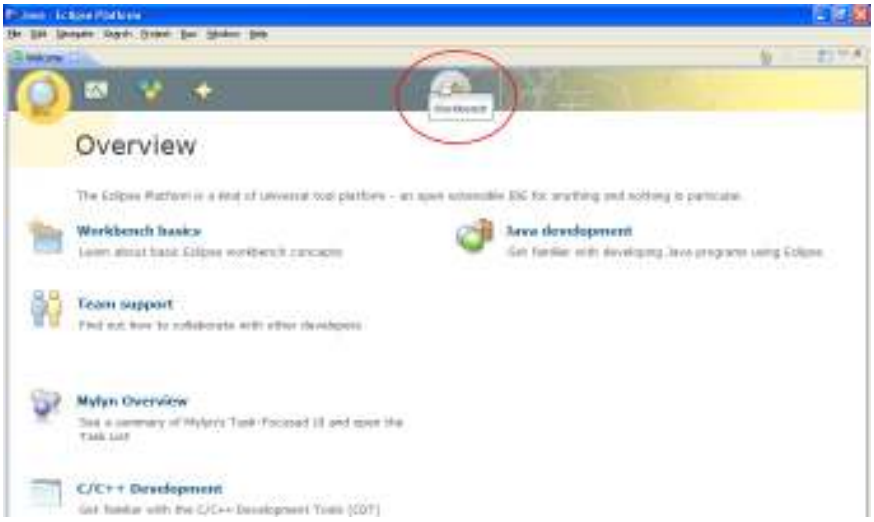
Usando o Eclipse pela primeira vez

Para executar o Eclipse, basta executar o arquivo "eclipse.exe", no diretório de instalação do Eclipse.

Uma tela perguntando pelo Workspace surgirá. O Workspace é o diretório no qual deseja armazenar seus projetos e suas configurações. Escolha o diretório que lhe convier (usuários do lig: recomenda-se escolher um diretório no drive Z, como "Z:\MeusProjetos\Eclipse"). Para que este Workspace seja sempre utilizado, ative o checkbox, como na figura abaixo.



Clicando em OK, o Eclipse exibirá uma tela de boas vindas. Clique na seta "Workbench" (figura abaixo) para sair da tela de boas vindas.



Você entrará automaticamente na "perspectiva" de trabalho em Java. Para mudar para a "perspectiva" de trabalho em C++, clique no menu Window → Open Perspective → Other. Selecione "C/C++" e clique em OK. Você estará agora na perspectiva de trabalho em C++.

Usando o Eclipse para programar em C/C++

Verifique no moodle o tutorial de uso do Eclipse para C/C++. Este tutorial (disponibilizado em breve) contém as etapas necessárias para se programar na perspectiva C++, dicas e erros comuns.

Dúvidas?

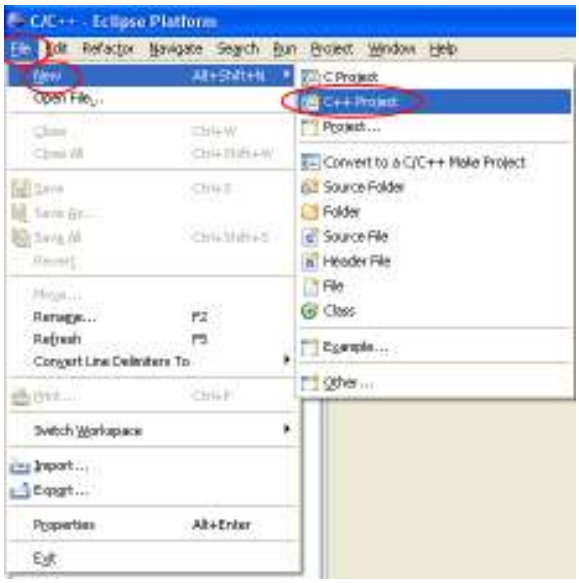
Os alunos que tiverem quaisquer dúvidas sobre o experimento, instalação ou configuração de software podem escrever um email para renan_montebelo@dc.ufscar.br. Verifique sempre se há novidades em http://moodle.dc.ufscar.br.

Usando o Eclipse para programar na linguagem C++

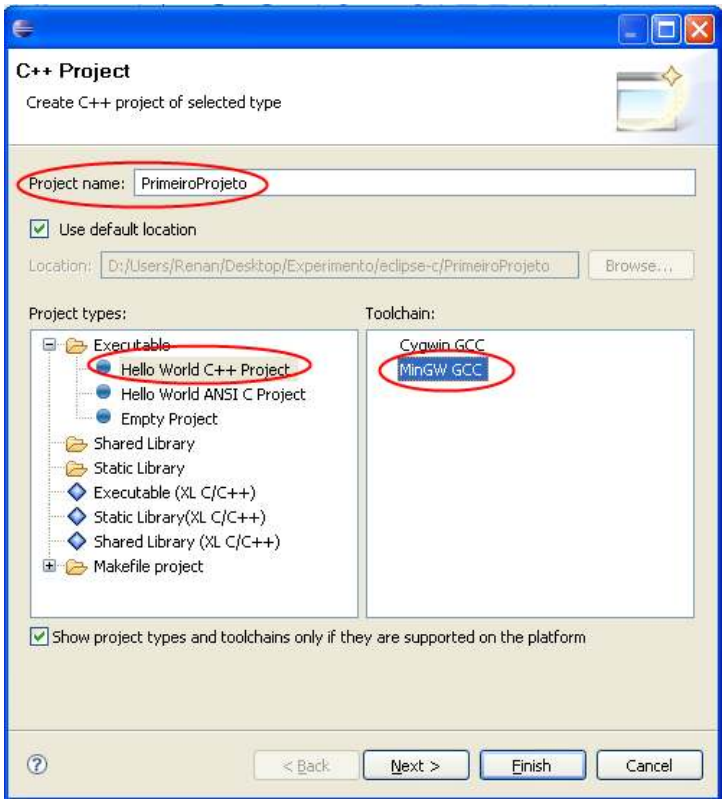
O objetivo desse documento é apresentar alguns exemplos de uso, recursos e erros comuns ao se programar na linguagem C++ com o Eclipse. Este tutorial considera que o Eclipse já está corretamente instalado e configurado. Para maiores informações, veja o Tutorial "Instalando e Configurando o Eclipse para C++" no Moodle (http://moodle.dc.ufscar.br).

Criando um novo projeto

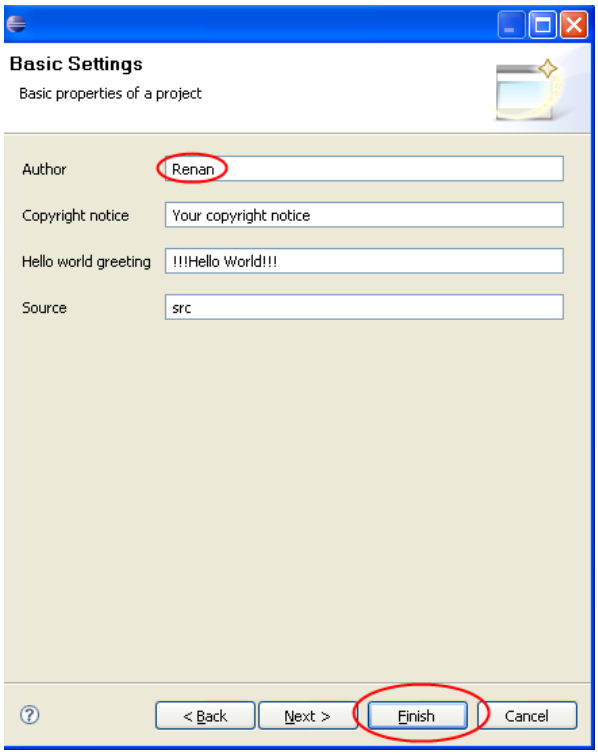
Para criar um novo projeto em C++, use o menu "File" → "New" → "C++ project".



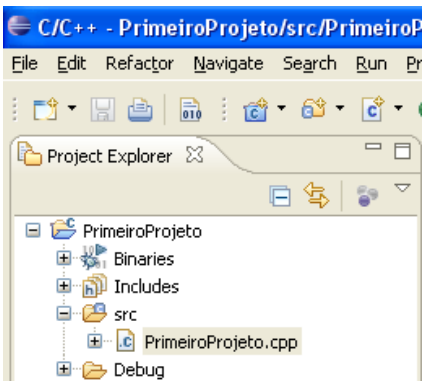
Insira um nome para o seu projeto (precisa ser um nome único para cada projeto Eclipse). Para o tipo de projeto, selecione "Executable" → "Hello World C++ Project". Para o compilador, escolha "Mingw Gcc". Mesmo para grandes projetos, escolha sempre o template "Hello World" pois assim o Eclipse também configura automaticamente algumas bibliotecas, o que evita muitos erros e problemas que algumas vezes são difíceis de detectar.



Clique em "Next". Na tela "Basic Settings" você pode colocar o seu nome (será usado nos comentários do programa, mas é opcional). Clique em "Finish".



Seu projeto está criado e é mostrado à esquerda. Ele deve conter 4 "pastas": Binaries, Includes, src e debug. A pasta que nos interessa é "src", que é onde todos os códigos-fonte devem estar. Expanda essa pasta e você encontrará um arquivo .cpp com o mesmo nome de seu projeto.

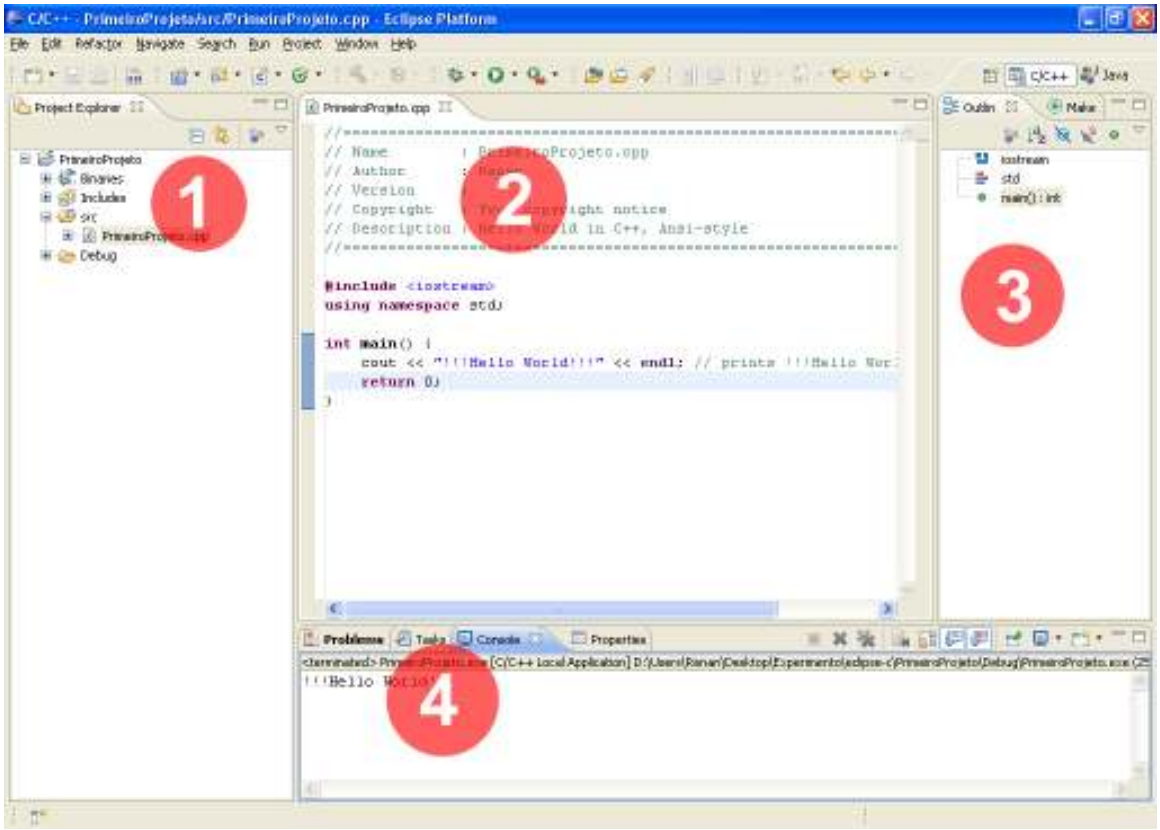


Ao clicar duas vezes (double-click) sobre esse arquivo, o mesmo será aberto no editor C++ do Eclipse.

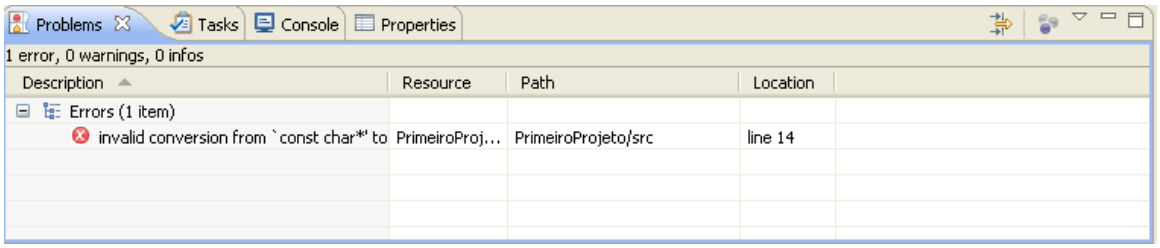
Vamos falar um pouco do ambiente de programação Eclipse. O ambiente do Eclipse é formado pelos seguintes componentes:

1. Project Explorer: aqui você pode acessar qualquer projeto seu, e para cada projeto você tem acesso a todos os arquivos. O Eclipse possui um editor inteligente, portanto se você clicar duas vezes sobre um arquivo-fonte, o arquivo será aberto em um editor C++; se você clicar sobre um arquivo .xml, um editor xml abrirá, e assim por diante.

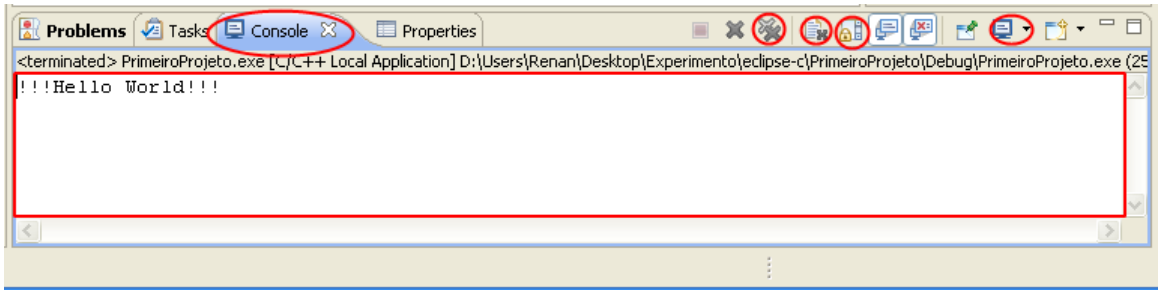
2. Editor: aqui é onde você edita o código. Você pode maximizar o tamanho do editor ao clicar duas vezes sobre o título do arquivo aberto, ou com o atalho Ctrl+M. Fazer a ação novamente minimiza o editor.
3. Outline. Aqui fica um "raio-x" do arquivo aberto, com acesso rápido a variáveis, includes, métodos, etc. No momento, o nosso projeto de "Hello World" é muito pequeno, mas conforme projetos crescem, o outline é muito útil para acessar rapidamente certas partes do arquivo aberto no editor.
4. Views. São pequenas "visões" do sistema. Explicaremos duas importantes a seguir.



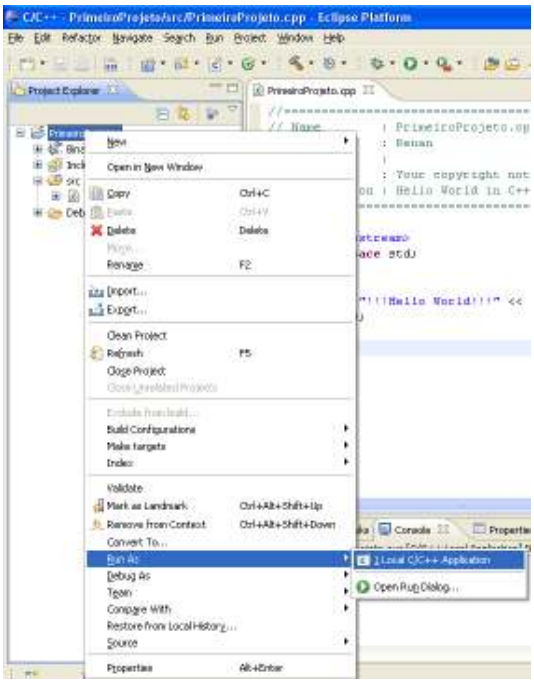
5. "Problems". Aqui são listados todos os problemas de compilação e avisos importantes sobre o código. Ao clicar duas vezes sobre o problema, o Eclipse automaticamente abre no editor o arquivo problemático, na linha problemática. Uma descrição do problema também está presente neste "view".



6. "Console". Aqui acontecem todas as entradas e saídas padrões de seu programa sendo executado. Vários "Consoles" podem ser abertos ao mesmo tempo, e ao final da execução de seu programa o console fica disponível para análise. Isso dispensa o uso do famoso System("pause") ao final dos programas C++. Na view "Console", o terceiro botão fecha todos os consoles de programas cujas execuções já terminaram, o quarto botão limpa o console atual, o quinto liga/desliga o auto-rolamento durante execução e o nono botão seleciona entre vários consoles abertos.



Para executar o seu programa, basta clicar com o botão direito sobre o nome do projeto → Run as... → Local C/C++ Application. Lembre-se de que toda saída e entrada serão executadas na view "Console".



Dicas

Algumas dicas de atalho úteis: ao acessar variáveis membro ou métodos de uma classe específica, ao pressionar ponto (“.”) ou (“->”), o Eclipse tem o recurso de auto-completar.

Alguns outros atalhos úteis são ativados ao usar as teclas Ctrl + Space simultaneamente. Exemplo: se você não se lembra exatamente da sintaxe do método main(), apenas digite main e aperte ctrl+ espaço simultaneamente que o Eclipse completará toda a sintaxe. O mesmo ocorre com comandos como for, do, if, switch, etc.

Um dos bons recursos do Eclipse é que aspas, parênteses e chaves são automaticamente fechados. Conforme você for digitando, note um “cursor especial verde”; ao teclar “enter”, o cursor de edição moverá automaticamente para o “cursor verde”.

É possível navegar entre os métodos de uma classe através de “links”. Exemplo: se existe uma chamada de método como “meuMetodo(x, y);”, basta clicar com o botão esquerdo do mouse enquanto pressiona a tecla Ctrl sobre “meuMetodo” que o Eclipse automaticamente vai para a declaração do método. O Eclipse possui “Avançar” e “Retroceder” entre os pontos de edição, como existem em um Browser. São duas setas amarelas situadas no canto superior direito.

É fácil exportar e importar um projeto no Eclipse. Para isso, clique com o botão direito do mouse sobre o nome do projeto → Export. Selecione no grupo “General” a

opção “Archive File”. Clique em Next. Na próxima tela, selecione todos os arquivos de seu projeto, escolha onde quer salvar o arquivo exportado (por padrão será salvo em zip). Recomenda-se ativar a opção “Create Directory Structure for Files”. Clique em Finish e seu projeto foi todo exportado.

Para importar um projeto é ainda mais fácil. Selecione no menu “File” → “Import”. Sob a categoria “General”, selecione “Existing Project into Workspace”. Selecione “Select Archive File”, clique em “Browse” e selecione o arquivo do projeto que deseja importar. Confirme os arquivos que deseja importar (provavelmente todos) e clique em Finish.

Eclipse Refactoring

Um dos pontos no qual o Eclipse se destaca sobre as demais IDEs é o refactoring. Ao clicar com o botão direito do mouse sobre o nome de uma classe, método ou variável, no menu há “Refactoring”. Podem haver várias opções de refactoring diferentes, sendo que a mais usada é “Rename”. Ao renomear uma classe ou método, o Eclipse renomeia em todos os arquivos, inclusive em chamadas de métodos e até mesmo comentários. Isso poupa bastante tempo e evita erros ao renomear manualmente.

Criando classes

Clique com o botão direito do mouse sobre a pasta “src”, selecione New → Class. Você deve informar o nome da classe. Opcionalmente você pode escolher quaisquer classes “base” da qual a nova classe deve herdar e se quer que o Eclipse já construa contrutores e destrutores padrões. Ao clicar em “Finish”, o Eclipse já vai construir o arquivo .cpp e o arquivo .h para você.

Dúvidas?

O Eclipse é um ambiente de desenvolvimento profissional, e há muito mais a ser explorado. É possível instalar plugins que ajudam durante o desenvolvimento. Há plugins que checam todo o código por erros comuns de programação, plugins que

permitem o desenvolvimento de interface gráfica, etc. O objetivo deste tutorial foi apresentar o básico de uso do Eclipse.

Os alunos que tiverem quaisquer dúvidas sobre o Eclipse podem escrever um email para renan_montebelo@dc.ufscar.br. Verifique sempre se há novidades em http://moodle.dc.ufscar.br.

Descrição dos procedimentos técnicos necessários para a participação no experimento sobre PSP

O objetivo deste documento é detalhar os procedimentos técnicos que os alunos devem seguir para participarem do experimento sobre PSP. A participação no experimento é voluntária e não influencia na nota final da disciplina (a menos que o professor responsável pela disciplina explicitamente diga o contrário). Os dados do experimento serão tratados de forma anônima.

Após a leitura do documento, os alunos podem se inscrever nas "disciplinas" do Moodle (**moodle.dc.ufscar.br**) referentes ao experimento. Ao se inscrever na disciplina, assume-se que o aluno é voluntário para participar do experimento e que o mesmo é capaz de cumprir as exigências deste documento. É importante que cada aluno se inscreva na "disciplina" do Moodle adequada à sua turma. As inscrições estarão abertas no período de 20 de agosto (segunda-feira) a 31 de agosto (sexta-feira). Todos os alunos possuem acesso ao Moodle do DC, sendo que login e senha são, inicialmente, o nome do usuário do domínio comp. Recomenda-se trocar a senha ao primeiro uso do Moodle; a senha do Moodle e do domínio comp são independentes.

Atenção: a ferramenta *Hackystat* é um software cliente-servidor que não funciona apropriadamente em redes com determinadas configurações de proxy ou firewall. No entanto, a ferramenta funciona corretamente nos LIGs do Departamento de Computação. Favor checar antecipadamente as configurações de sua rede. A ferramenta *Hackystat* utiliza o protocolo SOAP através da porta externa 80 (http). Também é importante destacar que a ferramenta armazena os dados pessoais de configuração no diretório "home" do usuário do Sistema Operacional; no Windows, esse diretório é "%userprofile%\.*Hackystat*". Portanto, a ferramenta pode não funcionar corretamente se várias pessoas compartilham um mesmo "login" do Sistema Operacional.

Pré-requisitos:

1. É necessário instalar o Java Runtime Environment. A versão mínima a ser instalada é a JRE 5, no entanto recomenda-se instalar a última versão disponível (JRE 6). Para download e instruções, acessar www.java.com. Nos LIGs, o Java 6 já está instalado em "C:\Arquivos de programas\Java\jre1.6.0_02"
2. Instalar a ferramenta Ant. Para download e instruções, acessar http://ant.apache.org/. No LIG, instalar o ant em "C:\Documents and Settings\NomeDoUsuário\apache-ant-1.7.0"
3. Configurar as variáveis de ambiente JAVA_HOME e ANT_HOME. Para instruções, visitar http://ant.apache.org/manual/index.html. Nos LIGs, a forma mais fácil de configurar as variáveis é pressionar simultaneamente as teclas "Windows" e "Pause/Break" do teclado, ir na *tab* "Avançado", clicar no botão "Variáveis de Ambiente" e adicionar as variáveis de ambiente no espaço do Usuário, e não do Sistema. Cada uma das variáveis deve apontar para o diretório de instalação do programa, por exemplo, JAVA_HOME deve apontar para "C:\Arquivos de programas\Java\jdk1.6.0_02" e ANT_HOME para "C:\Documents and Settings\NomeDoUsuário\apache-ant-1.7.0".
4. Adicionar nas variáveis de ambiente do usuário uma variável chamada PATH que contém os diretórios bin do Java e do Ant. Exemplo: "C:\Arquivos de programas\Java\jre1.6.0_02\bin;C:\Documents and Settings\NomeDoUsuário\apache-ant-1.7.0" (sem aspas, separados somente por ponto-e-vírgula). Não há problema se já existir uma variável PATH no espaço do sistema. Caso já exista PATH no espaço do usuário, adicionar os diretórios ao final da string que já existe, lembrando de deixar separado por ponto-e-vírgula.
5. Instalar a ferramenta LOCC. Para download e instruções, visitar http://csdl.ics.hawaii.edu/Tools/LOCC/. No LIG, instalar em "C:\Documents and Settings\NomeDoUsuário\locc-5.1".
6. Configurar a variável de ambiente LOCC_HOME para apontar para o diretório de instalação da ferramenta LOCC, analogamente ao passo 3.

7. Atualmente somente a IDE Eclipse é totalmente suportada pela ferramenta. O VisualStudio 2005, embora suportado pela ferramenta *Hackystat*, tem problema de conflito entre versões do "Microsoft .Net Framework". No LIG, é possível instalar o Eclipse no diretório "C:\Documents and Settings\NomeDoUsuário\eclipse. Assim, a instalação do Eclipse (e das demais ferramentas ali instaladas como Ant e Locc) acompanham o seu perfil quando realizar-se o login em qualquer máquina do LIG. O Eclipse pode ser baixado de www.eclipse.org, e existem versões prontas para Java ou C++.
8. Caso o aluno instale os softwares no LIG no diretório %userprofile%, é normal a lentidão ao realizar login / logout nos laboratórios, pois todo o seu perfil será copiado para a máquina local.

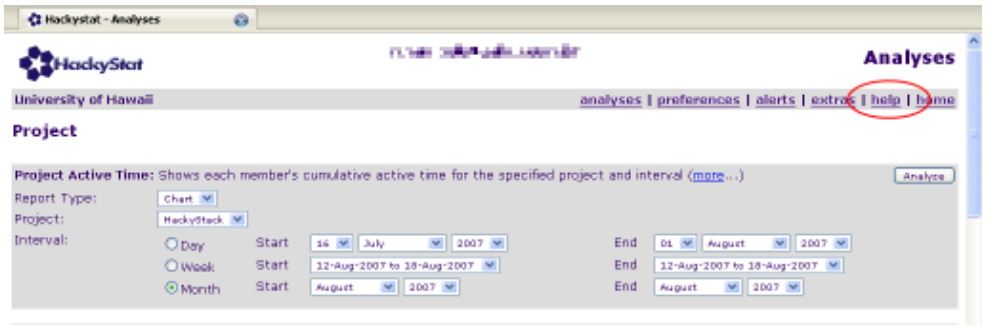
1º Passo: Registrar-se no servidor público

1. Acesse http://*Hackystat*.ics.hawaii.edu/
2. Entre com o seu endereço de email no campo "Email" e pressione o botão "Register".

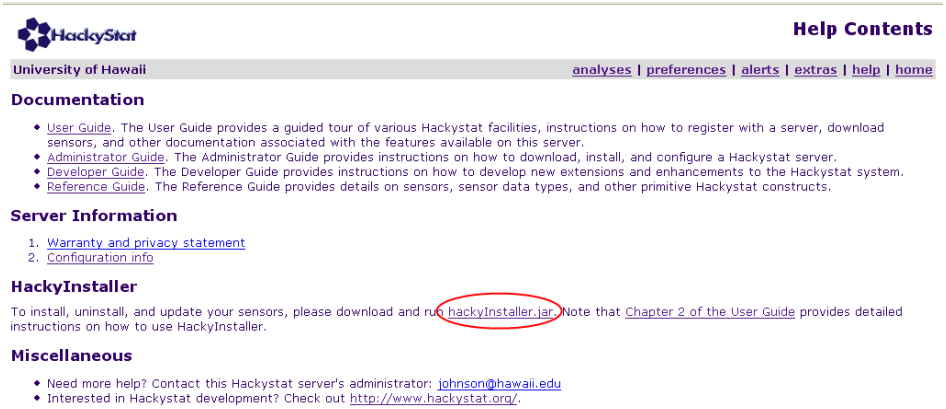
3. Um email contendo sua chave pessoal de acesso será enviado ao endereço de email fornecido. Favor checar sistemas anti-spam (liberar para johnson@hawaii.edu)
4. Entre com a sua chave pessoal no campo "Login" e pressione o botão "Login".

2º Passo: Instalar Ant e LOCC

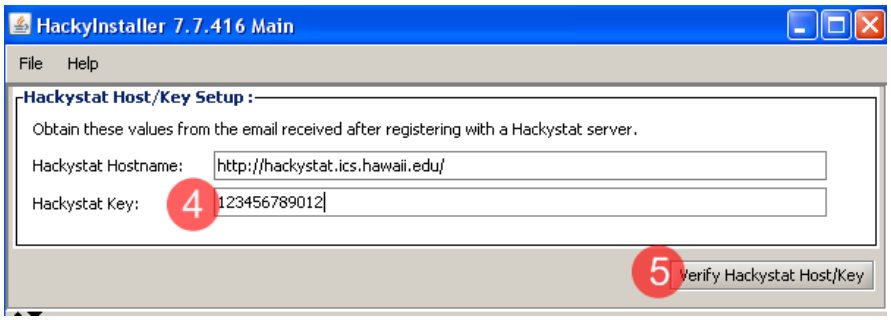
1. Uma vez feito o login no servidor público, acessar o link "Help".



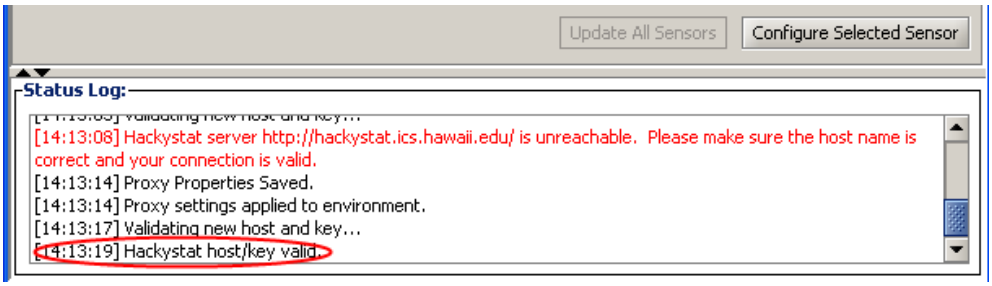
2. Clicar em "hackyInstaller.jar". Fazer o download do arquivo e salvá-lo em local de sua preferência.



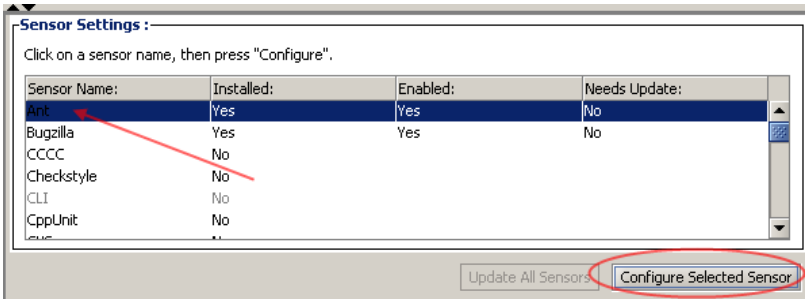
3. Executar o arquivo hackyInstaller.jar baixado (geralmente dois cliques funcionam; caso não funcione, usar o comando java –jar hackyInstaller.jar)
4. O programa instalador será executado. Colocar sua chave pessoal no campo "*Hackystat* Key"
5. Clicar no botão Verify *Hackystat* Host / Key



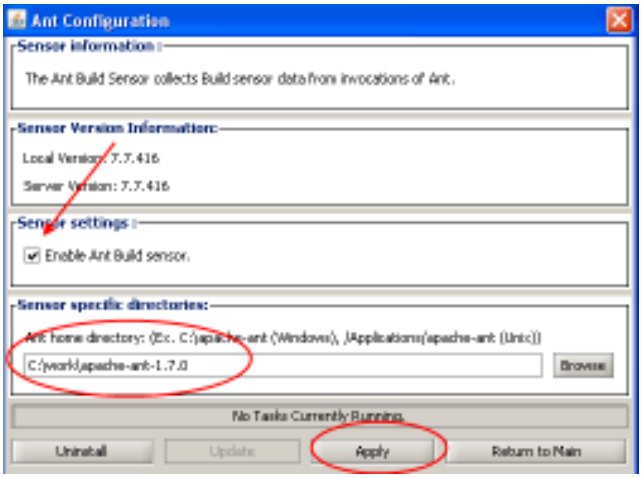
6. Verificar se a chave foi aceita (raramente não funciona direito, tentar novamente nesse caso). Se o status for *"Hackystat host/key valid."* significa que a chave foi aceita e tudo está corretamente funcionando. Caso o status seja *"Hackystat server http://Hackystat.ics.hawaii.edu/ is unreachable. Please make sure the host name is correct and your connection is valid."* pode siginificar um erro de comunicação com o servidor causado por firewall, proxy ou NAT. É essencial que o status indique que a chave pessoal é válida antes de continuar este tutorial.



7. Procurar na lista de sensores por “Ant”. Clicar em “Configure Selected Sensor”



8. Habilitar a opção “Enable Ant Build Sensor”. Digitar (ou usar a opção “Browse”) o diretório de instalação do Ant (ANT_HOME) no campo indicado. Clicar em “Apply”.

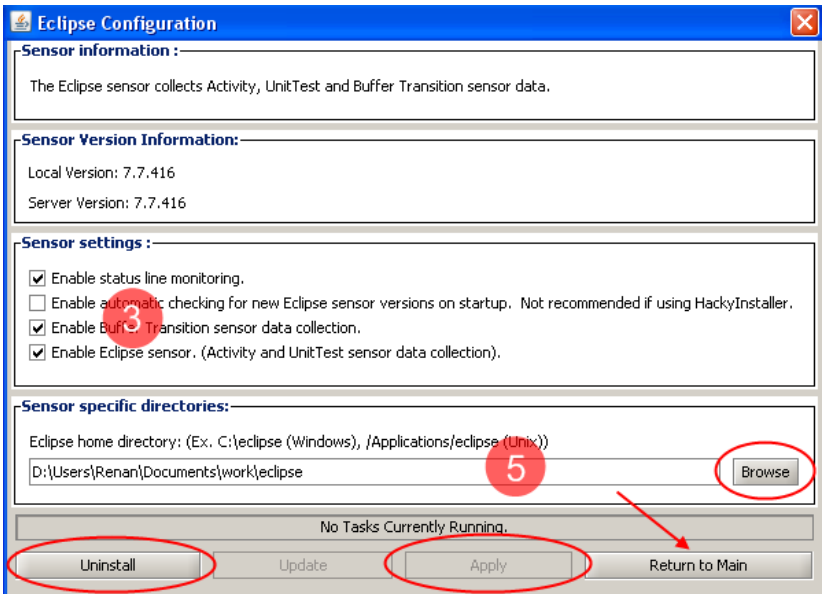


9. Após clicar em “Apply”, o botão “Install” será habilitado. Ao clicar em “Install”, o instalador fará o download do sensor para Ant e então configurá-lo. Clicar em “Return to Main” para voltar para o menu principal da ferramenta.
10. Realizar os passos 7, 8 e 9 para o sensor “Locc”. Os passos são análogos, no entanto, deve-se ter a atenção de fornecer o diretório onde o **Ant** está instalado, e **não** o diretório onde a ferramenta Locc está instalada.

3º Passo: Instalar o sensor do Eclipse

1. Selecionar na lista de sensores por “Eclipse”.

2. Clicar em “Configure Sensor”.



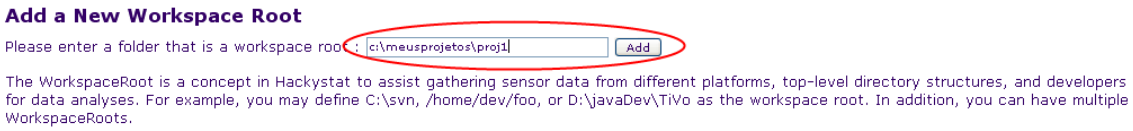
3. Habilitar as seguintes opções: “Enable Status Line Monitoring”, “Enable Buffer Transition sensor data collection.” e “Enable Eclipse Sensor”.
4. Não habilitar a opção “Enable automatic checking for new Eclipse sensor versions os startup”.
5. Digitar (ou usar o botão “Browse”) o endereço de instalação do Eclipse. Clicar em “Apply”. Clicar em “Install”. O instalador vai fazer o download e instalar o sensor para Eclipse. Clicar em “Return to Main”.

4º Passo: Criar projetos

1. Para cada novo projeto que você for realizar, deve-se criar um projeto no servidor.
2. Uma vez feito o login no servidor, clicar em “Preferences”.

3. Procurar pela seção Workspace. Clicar em “Go”.

4. Coloque o endereço de seu projeto (por ex: Z:\MeusProjetos\Proj1 ou /home/dev/proj1) e clique em "Add".



5. Clicar em "Preferences" novamente. Procurar a seção "Register Project" e clicar em "Register"

6. Preencher os dados de seu projeto, respectivamente:
    a. Nome do Projeto
    b. Data de início do projeto
    c. Data do término do projeto. Se o projeto não tiver uma data limite para término, selecionar "Undetermined".
    d. O email dos participantes. Neste caso específico, somente o seu email.
    e. O diretório "raiz" do projeto, como cadastrado na seção Workspace.
    f. Uma descrição geral do projeto.
    g. Clicar em "Create".

5º Passo: Usar o processo PSP

Uma vez configurado, seu ambiente será utilizado para utilizar o PSP. Dependendo de sua turma, serão necessárias diferentes tarefas. Mais detalhes serão divulgados no Moodle ao final do período de inscrições. Em geral, será necessário fazer algumas estimativas antes de cada projeto, como número de linhas, tempo necessário para concluir o projeto (em minutos) e número de defeitos (e tipos de defeitos) esperados ao término do projeto.

Como suporte a essas atividades, serão fornecidas algumas planilhas de Excel, arquivos “build.xml” prontos para uso e checklists de erros. No entanto maiores detalhes serão fornecidos somente ao final do término de inscrições das turmas.

Dúvidas?

Os alunos que tiverem quaisquer dúvidas sobre o experimento, instalação ou configuração de software podem escrever um email para renan_montebelo@dc.ufscar.br.

Personal Software Process - PSP

O PSP é um processo de melhoria contínua da qualidade de desenvolvimento de software criado para ser aplicado individualmente nos profissionais dessa área. Visto que aproximadamente 70% do custo do desenvolvimento do software estão associados aos esforços individuais no trabalho, as habilidades e as práticas adotadas por esses profissionais influenciam amplamente os resultados obtidos no processo de desenvolvimento. Torna-se imprescindível a utilização de um processo de melhoria que não apenas diga “o quê” deve ser feito, mas procure informar “como” alcançar a qualidade individual desejada.

Suas práticas são derivadas do CMM (Capability Maturity Model), desenvolvido pelo SEI (Software Engineering Institute). O CMM captura as melhores práticas de desenvolvimento de software utilizadas por grandes corporações e é organizado em 5 níveis de maturidade. O PSP é o resultado de uma adaptação do CMM projetado para atingir até o CMM nível 5, mas em foco individual, constituído por sete níveis.

Esses sete níveis do PSP são constituídos de formulários, relatórios, práticas e conceitos que evoluem a cada nível, tornando o processo de melhoria gradual e sem grandes impactos no desenvolvedor. A princípio, o livro-texto do PSP [Humphrey, 1995b] sugere que o processo descrito nele próprio deva ser seguido antes que o desenvolvedor realize as personalizações desejadas. O livro-texto sugere que as atividades de desenvolvimento de software sejam divididas em Planejamento, Desenvolvimento (projeto, codificação e testes) e Análise Final. Anterior ao planejamento, assume-se que o levantamento de requisitos tenha sido realizado e esse artefato sirva como entrada à primeira etapa mencionada.

Em Planejamento, todas as estimativas e predições são realizadas e o desenvolvedor concentra-se em produzir valores que se aproximem ao máximo dos futuros dados reais. Em Desenvolvimento, o profissional despende a maior parte do tempo efetivamente construindo o software, ou seja, seu objetivo é projetar, codificar e testar o software. Por último, como já diz o próprio nome, a Análise Final serve para levantar informações a respeito de todo o trabalho desempenhado nas fases anteriores e estabelecer um comparativo entre que foi planejado e estimado com o que foi na realidade executado. Segue uma breve descrição dos sete níveis do PSP.

- PSP Nível 0 - BASELINE

Esse nível representa apenas uma introdução do PSP nas práticas de desenvolvimento do profissional. Suas práticas são abrangentes o suficiente para que o desenvolvedor sinta pouca diferença entre o processo de desenvolvimento já em uso e o que está sendo introduzido. Algumas métricas começam a serem coletadas para que os alicerces do processo de melhoria sejam estabelecidos rapidamente, como o tempo gasto e defeitos identificados por fase de desenvolvimento.

Mesmo que as estimativas e as avaliações mais complexas ainda não estejam presentes nesse nível, a coleta dos dados é extremamente importante para que os cálculos estatísticos que serão utilizados mais adiante sejam válidos e representem efetivamente a evolução do profissional desde o início da utilização do PSP.

- PSP Nível 0.1

O nível 0.1 acrescenta os conceitos de padronização de código e de contagem de linhas de código, além de permitir que o desenvolvedor possa emitir propostas de melhoria para o processo. Essa padronização do código faz com que a contagem de linhas de código seja um valor consistente no decorrer do desenvolvimento dos programas, já que a inexistência desse padrão poderia gerar enormes diferenças na contagem.

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 e então inicia a fase de desenvolvimento onde são contadas as linhas de código geradas e o tempo despendido. Cada defeito identificado durante o processo é registrado e armazenado em um *Log* de Defeitos, para que se consiga identificar exatamente as maiores dificuldades. Um resumo do processo é mantido em todos os níveis contendo os dados mais importantes do desenvolvimento.

- PSP Nível 1 - PLANEJAMENTO

O nível 1 acrescenta atividades de planejamento ao PSP. 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 de linhas de código produzidas por tempo despendido desenvolvendo o trabalho, ao mesmo tempo que entendem a planejar e avaliar seu trabalho com os dados coletados.

Nesse nível do PSP, uma estimativa não é simplesmente calculada da forma que o desenvolvedor preferir. Introduz-se um método de estimativa para que o profissional tenha fundamentado uma maneira constante de estimar linhas de código entre os diferentes níveis [Humphrey, 1995b]. São acrescidos, portanto, um formulário para o armazenamento dessas estimativas e outro formulário voltado para a descrição e resultados dos testes a serem realizados no software. A introdução desse conceito de teste é muito importante para que a qualidade comece a ser foco do desenvolvimento do software porque é com eles que se pode garantir que um programa realmente desempenha as funções esperadas, evitando que um defeito seja notado apenas na fase de manutenção.

- PSP Nível 1.1

O objetivo desse nível é acrescentar o planejamento de recursos (principalmente tempo), a comparação do desempenho do desenvolvedor frente suas estimativas e os formulários

de planejamento de tempo e calendário. Isso significa que, a partir deste 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.

Os novos formulários introduzidos permitem ao desenvolvedor organizar as tarefas a serem realizadas e o seu tempo disponível para realizá-las. Dessa forma, é possível organizar o trabalho e medi-lo freqüentemente ao longo do de todo ciclo de desenvolvimento. Isso mostra ao profissional como é o desempenho de seu trabalho em cada uma das fases do processo, possibilitando que se note os pontos fracos e fortes, e que assim o desenvolvedor procure aprimorar os aspectos que representam maiores problemas para o seu desempenho pessoal.

- PSP Nível 2 - QUALIDADE

A qualidade passa a ser o principal critério dos conceitos a serem acrescentados. O ponto mais importante desse nível é a introdução de *checklists* de revisão usados para a identificação precoce de defeitos. Inicialmente, esses *checklists* são oferecidos prontos pelo livro-texto, mas o desenvolvedor é encorajado e responsável por analisar seus pontos fracos no desenvolvimento e assim personalizar esses *checklists*. Assim, é possível evitar a identificação tardia de seus erros, tanto de projeto quanto de codificação.

- PSP Nível 2.1

Esse nível passa a se preocupar não 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 projetos. A princípio, quatro propostas de modelagem são oferecidas pelo PSP e os critérios de garantia de qualidade são colocados no *checklist* de revisão de projeto. Assim, logo após o término da fase de projeto, o desenvolvedor usa o *checklist* como forma de identificar defeitos e melhorar seu projeto antes que a fase de Desenvolvimento seja iniciada.

Obviamente, o PSP não obriga que os modelos oferecidos para especificação do software sejam utilizados. O profissional pode utilizar o conceito proposto com qualquer modelagem que estiver sendo usada. Um exemplo seria utilizar UML e acrescentar nos *checklists* de revisão os principais erros introduzidos nos diagramas usados na modelagem.

- PSP Nível 3 - DESENVOLVIMENTO CÍCLICO

Até os níveis anteriores, o PSP é praticável para programas com até algumas poucas mil linhas de código. Seria 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.

Para isso, o PSP propõe que uma porção principal do software ("*kernel*") 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 e garante-se que uma outra iteração pode ser realizada sem que haja

regressão no desenvolvimento. O nível 3 é proposto com modificações um pouco mais profundas na estrutura do processo de desenvolvimento.

Se antes tínhamos um processo baseado em Planejamento, Desenvolvimento (projeto, codificação, compilação e testes) e Análise Final, agora temos: Planejamento geral, Projeto em alto nível, Revisão de Projeto em alto nível, Desenvolvimento Cíclico, Análise final e Integração. Dentro da fase de Desenvolvimento Cíclico, temos um processo semelhante ao desempenhado nos níveis anteriores do PSP: Projeto detalhado, Revisão do Projeto detalhado, Desenvolvimento de Testes, Codificação e Revisão do Código, Compilação, Testes e Reavaliação do Ciclo.

Em resumo, temos que o processo antigo é estabelecido em dois níveis distintos. O primeiro, direcionado ao desenvolvimento do software em nível de abstração maior, e o segundo, direcionado a cada ciclo a ser desenvolvido como aplicação dos níveis 2 e 2.1 do PSP.

O Experimento Proposto

O experimento aqui proposto procura avaliar, entre outros fatores, o grau de maturidade ideal para o primeiro contato com o PSP no contexto de um curso de graduação em Computação. Também procuramos determinar se a ferramenta de apoio *Hackystat* efetivamente ajudam na aplicação do método.

Algumas atividades do PSP não são suportadas pela ferramenta atualmente e, por esse motivo, não serão executadas durante o experimento. Também procuramos não exercer atividades que modificassem demais as atividades normalmente executadas nas disciplinas, como Testes Unitários com JUnit. Portanto os alunos terão contato com a abordagem PSP ∩ *Hackystat* ∩ Atividades da disciplina. Assim, teremos um subconjunto das atividades do PSP.

As principais atividades a serem realizadas neste experimento são:

- Nível 0 / 0.1
    - Planejamento do tempo total de desenvolvimento;
    - Registro de Defeitos Graves (adaptado do PSP);
- Nível 1 / 1.1
    - Uso de dados históricos para prever projetos futuros;
    - Previsão de Linhas de Código por Hora;
    - Tempo de da Fase de Testes;
    - Tentativa de elaborar um cronograma;
- Nível 2 / 2.1
    - Estimativa de defeitos graves (adaptado do PSP);
    - Número de defeitos por Mil Linhas de Código
    - Revisão de Código com Checklists

O Nível 3 não estará presente pois é o nível 2.1 aplicado ciclicamente para grandes projetos.

Para dúvidas sobre o PSP, acessem o Moodle (http://moodle.dc.ufscar.br) ou então enviem email para renan_montebelo@dc.ufscar.br.

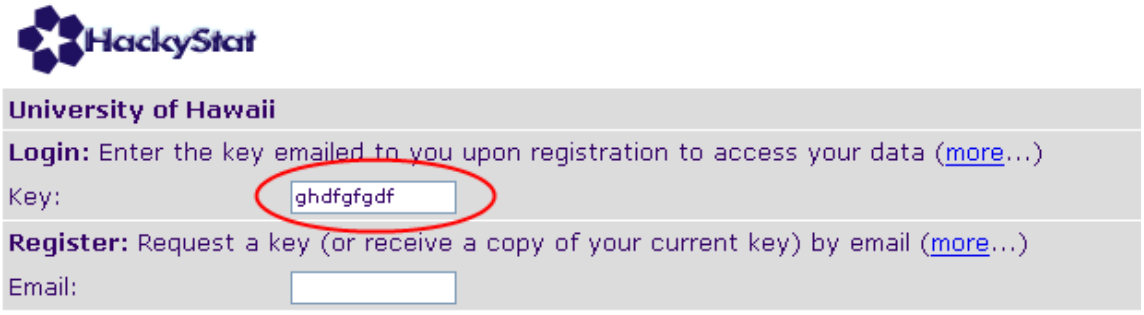
Exemplo de Utilização do PSP no Programa de Truco

PS: a planilha disponível no Moodle está preenchida com um exemplo de programa de truco. Os valores desta planilha são fictícios e absurdos, portanto não os usem como referência.

1) Baixe a planilha de Excel disponível no Moodle. A planilha chama-se Base_Historica.xls.
2) Preencha os dados básicos de seu programa como nome, disciplina e linguagem.
3) Na mesma planilha, estime o número total de linhas de seu programa. Estime também o tempo total de desenvolvimento do programa em minutos.

	A	B	C	D	E	F	G
1	**Número**	1					
2	**Nome**	Truco					
3	**Disciplina**	PC					
4	**Linguagem**	C++					
5	**Estimativa LOC**	350					
6	**Estimativa Minutos**	420					
7	**Estimativa de Defeitos**						
8	**LOC Real**						
9	**LOC Comentários**						
10	**Minutos Reais**						
11	**Defeitos Graves Removidos**						
12	**Tempo de Teste**						
13	**Tempo de Revisão**						
14	***Total LOC***	*0*	*0*	*0*	*0*	*0*	*0*
15	***Porcentagem Comentário / LOC***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
16	***Erro LOC***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
17	***Erro Minutos***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
18	***LOC / Hora***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
19	***Defeitos/KLOC***	*0,00*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
20	***Defeitos/Hora***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
21	**Tipos de Defeitos Comuns**						
22	**Palavras-Chave**						
23							

4) Instale o Eclipse e as ferramentas *Hackystat* como descritas nos manuais do Moodle.
5) Registre o seu projeto no servidor *Hackystat* como descrito no 4º passo do Manual "Como instalar o conjunto de ferramentas *Hackystat*.pdf" que se encontra no Moodle. Esse passo é essencial.
6) Usar normalmente o Eclipse para o desenvolvimento.
7) Ao final do desenvolvimento, acessar o servidor *Hackystat* e insira sua chave única de acesso.



8) Na página de Análises (como padrão é a primeira após o login), vá até a seção "Validation", onde a primeira opção deve ser "Active Time Trend".
9) Em "Report Type", selecione "Table". Selecione "Day" e insira a data inicial e final do desenvolvimento do seu programa. Clique em "Analyse".

10) Para cada dia de seu projeto, o servidor mostrará um valor em horas. Some as horas e as transforme em minutos (arredonde o valor final). Exemplo: 5,9 horas

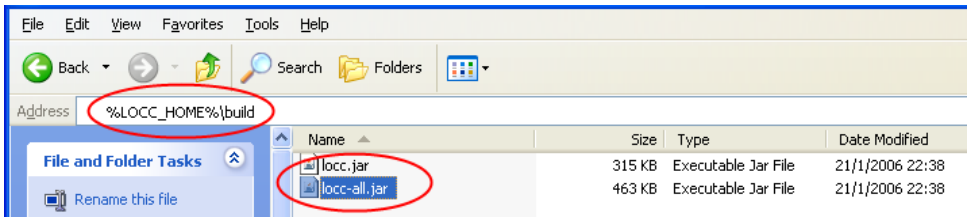
04-Sep-2007	05-Sep-2007
5.9	4.8

+ 4,8 horas = 10,7 horas ≈ 642 minutos.

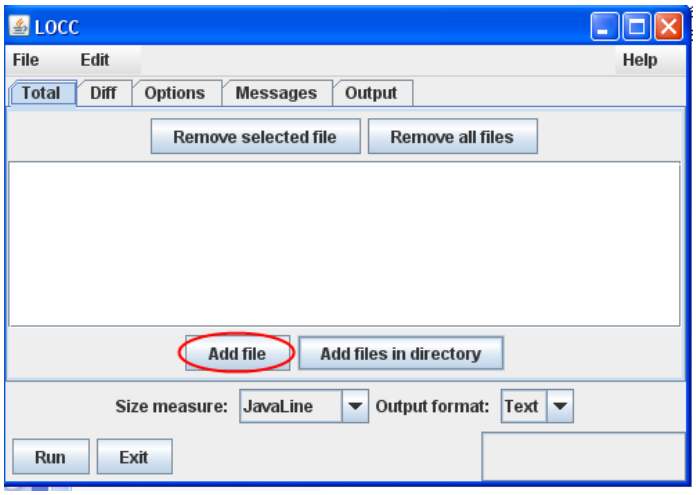
11) Insira esse valor no campo correto de seu arquivo Base_Historica.xls.

	A	B	C	D	E	F	G
1	**Número**	1					
2	**Nome**	Truco					
3	**Disciplina**	PC					
4	**Linguagem**	C++					
5	**Estimativa LOC**	350					
6	**Estimativa Minutos**	420					
7	**Estimativa de Defeitos**						
8	**LOC Real**						
9	**LOC Comentários**						
10	**Minutos Reais**	642					
11	**Defeitos Graves Removidos**						
12	**Tempo de Teste**						
13	**Tempo de Revisão**						
14	***Total LOC***	*0*	*0*	*0*	*0*	*0*	*0*
15	***Porcentagem Comentário / LOC***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
16	***Erro LOC***	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
17	***Erro Minutos***	*34,6%*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
18	***LOC / Hora***	*0,00*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
19	***Defeitos/KLOC***	*0,00*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
20	***Defeitos/Hora***	*0,00*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*	*#DIV/0!*
21	**Tipos de Defeitos Comuns**						
22	**Palavras-Chave**						
23							

12) Tenha certeza de que instalou a ferramenta LOCC. Vá até o diretório de instalação da ferramenta, procure pelo diretório "build" e execute (com dois cliques) o arquivo locc-all.jar.

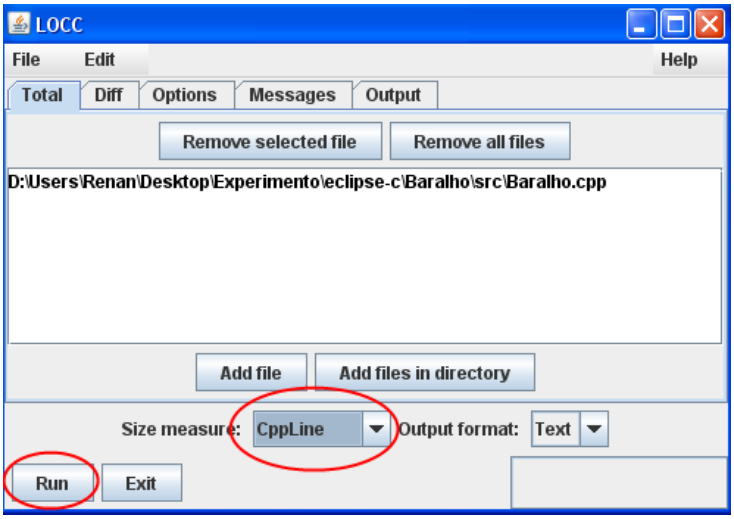


13) Selecione o botão “Add file”.

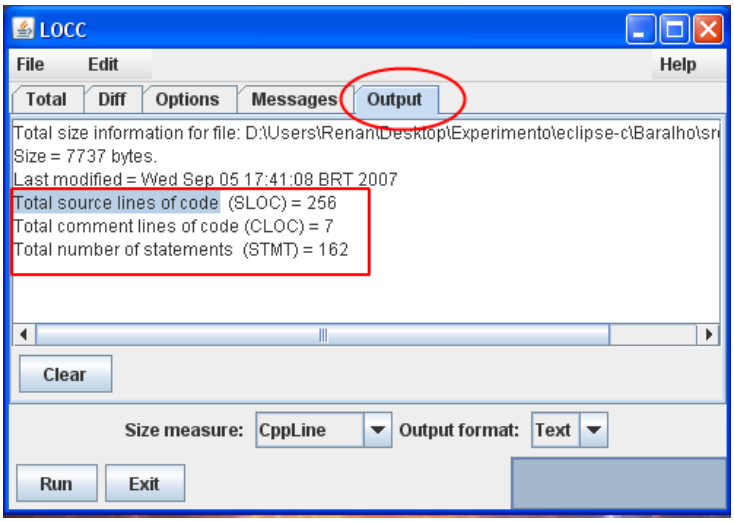


14) Selecione o seu arquivo fonte e clique em OK. Refaça para todos os arquivos-fonte de seu programa (ou alternativamente use o botÃo “Add Files In Directory”).

15) Selecione como medida “CppLine” e clique em “Run”



16) Ao clicar na aba “output”, o programa lhe indicará o total de linhas de comentário (“Total comment lines of code”) e o total de linhas de código-fonte (“Total source lines of code”). Entre com esses valores na planilha de dados históricos. Note que o total de linhas de código-fonte equivale a “LOC Real” na planilha de dados históricos.

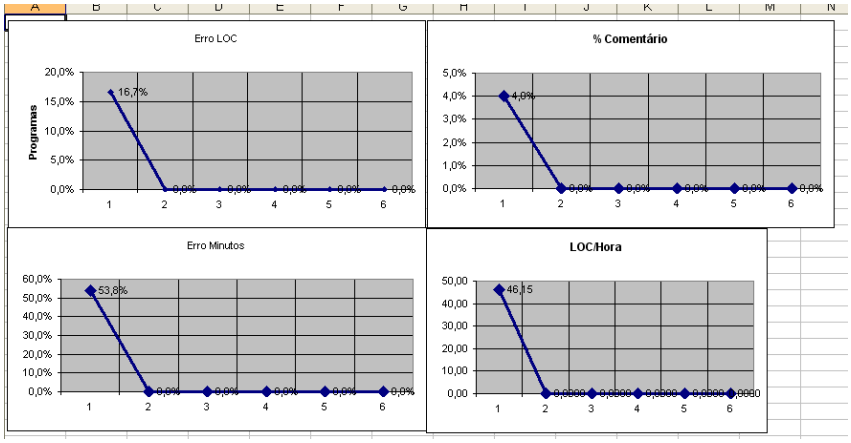


17) Termine de preencher a planilha com o número de defeitos graves removidos. Por defeitos graves entendam-se defeitos que foram difíceis de identificar e remover durante e fase de compilação e testes. Exemplo: se um programa entrou em loop infinito muitas vezes, conte como um defeito grave; se houve dificuldade no acesso com índices a dados de vetores (constantemente dando erro de memória), conte esse como um novo erro.

18) Analogamente ao passo anterior, preencha os tipos de defeitos graves na planilha de Excel.

	A	B
1	**Número**	1
2	**Nome**	Truco
3	**Disciplina**	PC
4	**Linguagem**	C++
5	**Estimativa LOC**	500
6	**Estimativa Minutos**	360
7	**Estimativa de Defeitos**	
8	**LOC Real**	600
9	**LOC Comentários**	25
10	**Minutos Reais**	780
11	**Defeitos Graves Removidos**	2
12	**Tempo de Teste**	
13	**Tempo de Revisão**	
14	***Total LOC***	*625*
15	***Porcentagem Comentário / LOC***	*4,0%*
16	***Erro LOC***	*16,7%*
17	***Erro Minutos***	*53,8%*
18	***LOC / Hora***	*46,15*
19	***Defeitos/KLOC***	*4,00*
20	***Defeitos/Hora***	*0,15*
21	**Tipos de Defeitos Comuns**	loop infinito de for(), acesso a vetores com índices fora de alcance
22	**Palavras-Chave**	truco, c++, rand, baralho, deck, shuffle
23		

19) Os campos em cinza da planilha são campos especiais de cálculos e não devem ser alterados. Na planilha 2 do próprio arquivo Base_Historica.xls você tem acesso a alguns gráficos prontos que auxiliam em sua própria auto-análise.



Dúvidas? Algum programa não funcionou como descrito nesse tutorial?

Envie email para: renan_montebelo@dc.ufscar.br