Competência pessoal em Qualidade de Software com PSP na sala de aula

Marcelo de Sousa Balbino[1], José Luís Braga[2]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

[2]Departamento de Informática – Universidade Federal de Viçosa (UFV)

marcelo@cientec.net, zeluis@dpi.ufv.br

Abstract. *To achieve high quality levels, software developers should be competent and acquire solid and sound technical knowledge. On the other side, software houses complain about the fact that the academic area does not provide the students and technicians with the necessary knowledge to climb to the high quality levels demanded by them. This paper describes a proposed and tested undergraduate Software Engineering program that provides the students with high quality knowledge and practice in the area. Our approach is partially based on PSP – Personal Software Process, and the results presented in this paper can be considered very good.*

Resumo. *Para alcançar o nível de qualidade necessário nos software de hoje é preciso primeiramente que o desenvolvedor desempenhe seu trabalho de forma competente. No entanto as industrias tem reclamado que o setor acadêmico não prepara os alunos para serem futuros profissionais da industria de software. Diante disso, o objetivo deste trabalho é relatar uma metodologia de ensino aplicada a disciplina de Engenharia de Software, que busca suprir as deficiências apontadas pelo setor industrial, através de um programa interdisciplinar e com a aplicação de técnicas voltadas para qualidade. O trabalho descreve principalmente a inclusão da técnica do PSP (Personal Software Process) no contexto da disciplina e os resultados alcançados com a aplicação desta técnica.*

1. Introdução

O software está cada vez mais presente na vida das pessoas, gerenciando desde aparelhos eletrodomésticos até equipamentos médicos, mísseis, aviões, telefones celulares, agendas pessoais, etc. O avanço tecnológico levou ao surgimento dos software embarcados que acompanham equipamentos, fazendo crescer a demanda por software de maior nível de complexidade e maior exigência de qualidade, principalmente nos equipamentos considerados críticos para a vida. A habilidade em lidar com o conceito de qualidade em software e, mais ainda, saber produzir software com determinado nível de qualidade passou a ser assunto obrigatório nos currículos dos cursos de computação.

Várias são as práticas que determinam um software de qualidade. Dentre elas estão um processo de gerência de projetos bem definido, um processo de gerência de recursos

humanos de qualidade, um processo de desenvolvimento de software adequado, dentre outros (Pressman, 2002). A comunidade de engenharia de software tem constantemente se dedicado ao desenvolvimento de técnicas cada vez mais aprimoradas, que auxiliem no alcance destas práticas.

No entanto, o que se tem observado nas organizações é que estas técnicas, quando usadas, são geralmente praticadas de forma inadequada. Isto porque muitas vezes as metodologias estão voltadas para o processo e o produto, e deixam de lado o elemento principal que é o executor da técnica. É preciso capacitar adequadamente os indivíduos desde as primeiras oportunidades, pois todas estas metodologias perdem o valor se o desenvolvedor de software não conhece os conceitos associados e não os aplica na sua rotina diária.

Para se desenvolver competências em qualidade de software no nível individual, é preciso antes de tudo encarar o desenvolvimento de software como um trabalho de engenharia. No entanto, a forma como a maioria dos desenvolvedores de software trabalha atualmente tem mais de arte do que de engenharia, imperando as competências pessoais em que cada um tem suas próprias adaptações de métodos e técnicas de desenvolvimento (Humphrey, 1995).

A prática de produção de software como trabalho usando técnicas das engenharias implica, primeiramente, em introduzir a disciplina e organização próprias da área das engenharias no desenvolvimento de software. O ensino puro e simples de boas técnicas de engenharia, quando se trata de profissionais já atuantes no mercado, não é a parte mais complicada. O desafio maior consiste em disciplinar e mudar o perfil deste profissional, principalmente no plano das práticas individuais. Esta mudança cultural do profissional é muito mais árdua do que no estudante que está em fase de desenvolvimento do seu perfil.

Defende-se neste trabalho que o desafio que se impõe às instituições de ensino reside em lapidar o perfil de seus estudantes para que estes ingressem no mercado com os conceitos de engenharia, qualidade e disciplina já fixados. A existência de um "vácuo" entre as exigências atuais de mercado na área de engenharia de software e a educação acadêmica correspondente tem sido amplamente registrada (Dion, 1993; Hilburn et al. 1995). A indústria requer engenheiros de software, mas as universidades têm formado cientistas da computação (Hilburn et al. 1995). Estudos recentes de um instituto de engenharia de software registrou sucintamente o estado do problema: "Os estudantes não estão preparados para fazer o salto da ciência da computação ou da engenharia de software na escola para a engenharia no mundo profissional" (Towhidnejad & Hilburn, 1997).

Um dos motivos da existência deste "vácuo" é a carência de programas acadêmicos voltados para a competência necessária nas tarefas da vida profissional. Dentre estas competências estão o desenvolvimento de grandes projetos, a habilidade de trabalhar em equipe, o conhecimento de processos, a qualidade e habilidade de planejar, estimar tempo e elaborar custos (Towhidnejad & Hilburn, 1997).

Este trabalho relata a experiência adquirida em uma disciplina introdutória de Engenharia de Software do Curso de Ciência da Computação da UFV, em que os conceitos de qualidade no nível pessoal são ensinados e praticados desde a primeira aula. Os alunos têm oportunidade de aplicar esse conhecimento em um projeto prático de sistema que se aproximam do mundo real, envolvendo as três dimensões principais do projeto de software:

interface, *controle* e *dados*. A técnica adotada é o PSP (Personal Software Process) (Humphrey, 1995) apenas em seus níveis de competência iniciais, o que se mostrou suficiente para uma conscientização dos alunos quanto ao tema, seus desafios e desdobramentos.

Com essa iniciativa, já foi possível percebe r uma mudança sensível na visão de produção de software dos alunos, principalmente no que diz respeito à importância dada aos aspectos de qualidade, e o cuidado dos alunos com os mesmos. Um ponto positivo do PSP no contexto acadêmico é que uma de suas características é ser uma técnica de auto-convencimento, fazendo com que os próprios alunos percebam a melhoria obtida ao empregar a técnica: "Com a aplicação do PSP é possível gerenciar melhor o tempo de desenvolvimento e diminuir o número de erros cometidos, o que certamente vai gerar um produto mais confiável e fácil de manter", um dos vários depoimentos de alunos colhidos nas vezes em que o tema foi oferecido.

No item 2 serão apresentados os principais características da técnica do PSP. No item 3, são apresentados aspectos gerais da metodologia empregada na disciplina de Engenharia de Software e no o item 4 os principais resultados alcançados pela aplicação desta metodologia. Por fim, no item 5 são traçadas as conclusões do trabalho.

2. Personal Software Process (PSP)

Em 1995, Watts S. Humphrey apresentou o Personal Software Process (Humphrey, 1995). O PSP é um processo que permite avaliar a competência e melhorar a qualidade individual no desenvolvimento de software, ajudando o desenvolvedor a controlar, admin istrar e aperfeiçoar sua forma de produzir software, oferecendo um modelo que organiza a maneira na qual cada indivíduo executa seu trabalho.

A filosofia por trás do PSP é que a competência de uma organização para construir software de tamanho e complexida de razoáveis decorrem em parte da habilidade individual de seus engenheiros de software para desenvolver, com alta qualidade, pequenos programas de maneira disciplinada e efetiva (Towhidnejad & Hilburn, 1997). Tanto o CMM (Capability Maturity Model) quanto o PSP se baseiam no princípio do conhecimento, avaliação e melhoria continuas do processo de desenvolvimento. Enquanto o CMM se concentra na melhoria da capacidade organizacional, o PSP objetiva a melhoria do desenvolvedor de um ponto de vista individual (Humphrey, 1995).

O modelo de competência organizacional do CMM baseia-se em cinco níveis e dezoito KPAs (Key Process Areas), distribuídas pelos cinco níveis. Cada KPA representa uma componente fundamental do processo, sendo desdobrável em metas e essas em atividades. Existe uma ligação estreita e intencional entre as KPAs do CMM e a melhoria no nível pessoal proporcionada pelo PSP. Doze das dezoito KPAs são diretamente favorecidas pelas práticas do PSP (Humphrey, 1996). A mensagem é clara: fomentar a competência pessoal com aplicação do PSP afeta em proporção direta a competência organizacional, fato que é explorado pelas pequenas empresas que não têm recursos para aplicar na migração para níveis altos do CMM.

Resumidamente, o PSP proporciona (Humphrey, 1 995):

- um processo de desenvolvimento básico, seqüencial ou em cascata, que serve como ponto de partida. Se o profissional for experiente e já tiver algum processo definido que prefira seguir, não há impedimento;
- uma estrutura para coleta de dados adaptável a qualquer processo de desenvolvimento. Os dados coletados se concentram no que é considerado o ponto principal da qualidade: diminuição do número de erros cometidos durante o desenvolvimento do software, e conhecimento das causas principais que levaram aos erros. Os erros devem ser rastreados e as fases onde são cometidos devem ser identificadas, para permitir análise posterior e conseqüente melhoria;
- coleta de dados a respeito do tempo gasto em cada fase do processo proposto. O tempo é segmentado em tempo em tarefa útil, tempo de interrupção por qualquer motivo que deve ser anotado juntamente com a classe de interrupção e finalmente o tempo gasto na detecção e correção dos erros encontrados.
- formulários específicos, denominados "logs", onde os dados são coletados. A cada projeto terminado, os dados são transferidos para um formulário de resumo de planejamento de projeto (Project Plan Summary) e totalizados com os dados dos projetos anteriores.
- seqüências de exercícios específicos, propostos para proporcionar avanço incremental no conhecimento do método e avaliação também incremental dos resultados alcançados, que se fazem sentir logo nos primeiros exercícios. São duas seqüências distintas, a série A com 10 exercícios, e a série B com 9 exercícios, sendo que qualquer uma das duas pode ser usada para treinamento, além de cinco relatórios de progresso e avaliação que devem ser preenchidos e analisados em pontos específicos das seqüências. Esses exercícios proporcionam a base de comparação e extração de dados, pois todos os programadores de um mesmo grupo são submetidos aos mesmos exercícios. Não há exigência quanto a linguagem de programação a ser usada. A exigência é que seja tudo feito de maneira sistemática, ou seja: uma vez escolhida a linguagem de programação, por exemplo, ela deve ser mantida até o final da bateria de exercícios.

Na medida em que se avança na aplicação do PSP, o processo de desenvolvimento pessoal fica mais conhecido, a produtividade em linhas de código por unidade de tempo pode ser calculada com base no histórico registrado no resumo de planejamento, a qualidade medida em erros cometidos por linha de código produzida e os principais motivos de interrupções, que fatalmente quebram o processo de criação, ficam evidentes. Com esse conhecimento básico, é possível alcançar outras metas, como por exemplo a competência em planejamento e conseqüentemente em determinação de custos de projetos de software e a melhoria do processo de desenvolvimento baseada em cálculos estatísticos sobre os dados coletados.

3. Aspectos metodológicos

A disciplina em que os conceitos de PSP são introduzidos é a Engenharia de Software I, situada no quinto período da grade curricular do Bacharelado em Ciência da Computação da UFV. No quarto período, os alunos cursam Bancos de Dados I, em que são ensinados os

conceitos básicos em bancos de dados e onde os alunos desenvolvem um projeto completo de bancos de dados, que é implementado em software disponível nos laboratórios do Departamento de Informática (DPI), juntamente com parte da aplicação correspondente. O foco do trabalho é no projeto e implementação do banco de dados, sendo que a parte funcional da aplicação, incluindo interfaces com usuários, é implementada apenas como complemento para que o trabalho fique visualizável, não sendo ensinadas técnicas de engenharia de software.

Na Engenharia de Software I, o mesmo problema proposto como motivação para o projeto em Bancos de Dados I é utilizado novamente como tema para o projeto que será desenvolvido no decorrer do semestre. As dimensões relativas a *interfaces* e a *controle* são então desenvolvidos dentro das técnicas da área, e implementados utilizando a dimensão *dados* já desenvolvido e implantado no semestre anterior. Essa sinergia entre as duas disciplinas tem se mostrado muito benéfica e eficaz do ponto de vista de aprendizado.

3.1 Organização do tópico na disciplina

A disciplina Engenharia de Software I tem seu conteúdo definido conforme o recomendado nas diretrizes curriculares da SBC - Sociedade Brasileira de Computação (http://www.sbc.org.br/). O grande desafio foi inserir o tópico "qualidade de software com PSP" nas aulas iniciais da disciplina, sem prejudicar o seu andamento normal.

A disciplina está sendo oferecida no presente semestre (2004/I), com base nas seguintes decisões originadas das experiências anteriores:

- o tempo dedicado ao tema não deve ultrapassar três semanas de aulas, o que dá um total de 12 horas/aula mais o tempo extra-classe para o desenvolvimento dos exercícios e preenchimento das planilhas;
- os exercícios escolhidos foram os três primeiros da série A, mais os relatórios R1, R2, R3 e R4. A seqüência de aplicação dos exercícios e a intercalação com as aulas explicativas sobre os temas pertinentes foi feita seguindo a sugestão do próprio Humphrey (1995);
- o nível máximo PSP que se deseja alcançar com essa prática é o PSP 1.1, que ainda não trata de competência em planejamento, mas já mostra o caminho para que se chegue a ele;
- os capítulos do livro (Humphrey, 1995) utilizados e necessários para cumprir esse objetivo foram os de 1 ao 4, mais parte do material do mesmo livro, onde estão descritos os exercícios, conteúdo de relatórios, etc;
- os formulários utilizados para coleta de dados foram: *Time Recording Log* para registrar o tempo gasto em cada fase; *Defect Recording Log*, para registro dos dados referentes a cada erro encontrado nas fases do processo e sua correção; Os dados destes dois formulários são lançados posteriormente no *Project Plan Summary* destinado ao planejamento das fases do processo básico, ao resumo e acumulação dos dados; Foi preparado um conjunto de planilhas em Excel para facilitar o preenchimento dos dados, seguindo fielmente o modelo utilizado no livro. Há software de apoio com licença "freeware" na Internet, como o PSPDashboard

(http://processdash.sourceforge.net/), mas preferiu-se utilizar as planilhas Excel que exigem mais atenção dos alunos, levando-os a pensar mais e melhor na qualidade das anotações que fizeram.

Os exercícios foram corrigidos na mesma semana em que foram entregues, sendo dado o necessário retorno de avaliação aos alunos a tempo de os possíveis erros e falhas de entendimento serem corrigidos antes do próximo exercício. Uma análise individual e do grupo de alunos foi feita, e os gráficos e conclusões resultantes foram levados à sala de aula e mostrados aos alunos. Parte dessas análises é mostrada na seção de Resultados e discussão, a seguir.

4 Resultados e discussão

A análise dos dados é desenvolvida em duas etapas. Primeiramente é feita uma verificação dos dados de cada aluno individualmente, o que convencionou-se chamar de "análise de consistência dos dados". Esta verificação inicial visa somente constatar se os dados foram anotados de forma completa, correta e consistente pelo aluno. Em uma segunda etapa, a que se deu o nome de "análise qualitativa" dos dados, é feita uma análise da prática de desenvolvimento de software de cada aluno, e os resultados são lançados em um gráfico para a turma como um todo.

4.1 Análise de consistência

A análise de consistência se baseia no princípio de que o aluno não deve fazer adaptações no processo nestes exercícios iniciais, o que é encorajado apenas quando se atinge o nível de competência adequado. O objetivo da análise de consistência é certificar se o estudante anotou nas planilhas todos os dados que o processo exige, mesmo que a princípio ele não veja motivo para tal. Um exemplo de uma situação muito comum é o aluno não fazer as estimativas de erros e tempo da primeira vez, por não ter experiência e parâmetros para tal. O recomendado é que seja estipulado um valor inicial para os parâmetros solicitados, apenas para dar início ao processo e para que o aluno perceba a dificuldade de realizar um planejamento sem ter um histórico de dados, o que valoriza ainda mais o exercício do PSP.

Baseado nas análises efetuadas, e nas principais falhas anotadas no preenchimento das planilhas e informações ao longo do tempo em que a técnica foi aplicada na sala de aula, chegou-se à Tabela 1, que mostra os principais pontos a serem verificados na avaliação. Essa tabela é um resultado específico desse trabalho, e não consta de nenhum material relativo ao PSP.

Como exemplo de uso das informações da Tabela 1, seja o caso do terceiro item da lista de verificação. Muitas vezes o aluno não faz o preenchimento individual dos erros no Defect Recording Log e, neste caso, apenas se pode apurar se o número de erros inseridos está consistente com os erros removidos. É aceitável até que o número de erros inseridos ultrapasse os removidos, pois pode acontecer de o aluno não conseguir retirar todos os erros do programa. No entanto, a situação inversa certamente mostra uma inconsistência que pode ter sido gerada pela desatenção no preenchimento dos dados. Outra situação importante a se observar na análise de consistência dos dados é se existem dados muito discrepantes anotados por algum aluno, o que pode ter sido causado por erro de

preenchimento, não entendimento das informações solicitadas nos formulários, desatenção causada por fatores externos, etc.

Tabela 1: Lista de verificações de consistência dos dados

Número	Descrição
1	O tempo gasto nas fases anotados no Time Recording Log e no Project Plan Summary estão iguais
2	Foi feito o planejamento de tempo e dos erros para cada fase do processo
3	Os valores anotados no Project Plan Summary para o número de erros injetados e removidos estão iguais
4	Número de erros anotados no Project Plan Summary igual ao número de erros descritos no Defect Recording Log
5	As fases dos erros descritos no Defect Recording Log conferem com o número de os erros inseridos e/ou removidos em cada fase no Project Plan Summary
6	As colunas Inject e Remove do Defect Recording Log foram usadas corretamente para anotar as fases onde os erros foram inseridos e removidos respectivamente
7	A coluna Fix Defect foi usada corretamente, ou seja, apenas quando um erro for inserido ao corrigir outro erro. A coluna indica qual erro se estava corrigindo quando este novo erro foi inserido
8	Os valores da coluna Delta Time no Time Recording Log foram calculados corretamente
9	Os dados do Time Recording Log e do Defect Recording Log foram copiados para o Project Plan Summary
10	A coluna Plan foi usada corretamente para o planejamento (estimativas do tempo e dos erros)

Fonte: resultados da pesquisa

A verificação individual de consistência dos dados é importante não só para dar o devido retorno ao aluno quanto ao correto preenchimento, mas também para possibilitar a análise coletiva da turma, que é feita em seguida. Na análise coletiva não devem ser incluídos os dados para os quais se tiver alguma suspeita de que estão incorretos, para não comprometer as conclusões.

4.2 Análise qualitativa

Na análise qualitativa dos dados procura-se analisar e conhecer o perfil de programação dos alunos, com a finalidade de instruí-los a desenvolverem seu trabalho de acordo com as boas práticas de desenvolvimento de software. Para atingir esse objetivo, a fase de análise qualitativa divide-se em duas subfases: a análise individual e a análise coletiva.

A análise individual consiste em extrair dos relatórios as informações sobre: a distribuição do tempo dedicado a cada fase; as fases onde os alunos cometem mais erros; onde estes são removidos e a precisão das estimativas de tempo e erros.

A análise da distribuição do tempo gasto em cada fase verifica se o aluno dedicou seu tempo de forma coerente e proporcional na implementação do programa. Com base na experiência do professor, na literatura, ou na própria média da turma deve-se determinar uma faixa de percentual de tempo aceitável para cada fase, para então verificar se o percentual de tempo gasto pelo aluno em cada fase se encontra dentro da faixa de

percentual aceitável. Agrupando-se os dados de todos os alunos e determinando-se a média dos percentuais de tempo em cada fase é possível realizar a mesma análise no âmbito coletivo.

Partindo-se do princípio de que quanto mais tarde um erro é encontrado mais cara é sua remoção ou correção, a idéia é que os eventuais erros sejam identificados o mais cedo e o mais próximo possível da fase onde foram introduzidos no sistema. Considera-se que o aluno fez um bom trabalho quando os erros são encontrados até a fase de revisão de código. Procura-se observar também se o estudante não cometeu um número exagerado de erros face ao nível de dificuldade do exercício.

Na anotação de tempos, a informação importante a ser extraída é a discrepância entre o tempo estimado e real por fase do processo. Para passar ao nível de competência em Planejamento, as estimativas têm que estar coerentes, e os alunos têm que ter apreendido muito claramente a necessidade de ter sempre em mãos os históricos anotados de projetos anteriores, que formarão a sua base de conhecimento individual em competência pessoal. A discrepância entre o estimado e o real foi medida usando a fórmula a seguir: $\%\_erro\_estimativa = (tempo\_estimado - tempo\_realizado)/tempo\_estimado$

4.3 Coleta de dados

Para registro e análise dos dados sobre a turma, foram usadas as planilhas, como a da Figura 1, que estão exemplificadas com os dados de 2 alunos da turma para o programa 1A. O mesmo procedimento foi repetido para os programas 2A e 3A. Os valores contidos nos trechos de planilhas abaixo são reais, apenas os nomes dos alunos são fictícios.

Uma das planilhas desenvolvida foi a planilha com observações de preenchimento dos dados do PSP (Figura 1) que indica os problemas detectados para a forma de preenchimento de cada aluno, segundo os critérios de verificação apresentados na Tabela 1 para um programa da série de exercícios. Segundo a figura, os dados do estudante Eduardo Silva não contemplaram o item 9 da lista de verificação (Tabela 1).

Matr.	Nome	1	2	3	4	5	6	7	8	9	10	Total
001	Estudante1	X	X		X				X			
002	Estudante2									X		
Dados dos demais alunos ...												
Número ocorrências		13	18	4	10	8	1	1	3	7	1	66
% por Tipo de Erro		19,70	27,27	6,06	15,15	12,12	1,52	1,52	4,55	10,61	1,52	

Fonte: resultados da pesquisa

Figura 1: Observações de preenchimento das planilhas do PSP

A planilha de percentual de tempo gasto em cada fase (Figura 2) indica como foi a distribuição do tempo gasto pelo aluno em cada fase do processo de desenvolvimento de software para um programa da série de exercícios.

Matr.	Nome	Planning	Design	Code	C. Review	Compile	Teste	Postm.
001	Estudante1	4,84	24,19	35,81	0,00	12,90	16,13	6,13
002	Estudante2	10,00	10,00	40,00	3,33	10,00	16,67	10,00
Dados dos demais alunos ...								
Média		11,52	9,53	41,82	6,02	10,64	13,09	7,38

Fonte: resultados da pesquisa

Figura 2: Percentual de tempo gasto em cada fase do processo

Já a planilha de percentual de erros removidos em cada fase (Figura 3) indica como foi a distribuição dos erros removidos pelo aluno em cada fase do processo de desenvolvimento de software para um programa da série de exercícios. A mesma planilha foi desenvolvida verificando as fases onde os erros foram inseridos.

Matr.	Nome	Planning	Design	Code	C. Review	Compile	Teste
001	Estudante1	0,00	0,00	12,50	12,50	75,00	0,00
002	Estudante2	0,00	0,00	0,00	0,00	50,00	50,00
Dados dos demais alunos ...							
Média		0,00	0,00	7,67	5,09	71,37	15,88

Fonte: resultados da pesquisa

Figura 3: Percentual de erros removidos em cada fase do processo

4.4 Análise coletiva

A partir dos dados coletados nas planilhas descritas no item anterior e com o auxílio de gráficos gerados também a partir destas planilhas é possível fazer uma análise das características da turma no que se refere à prática do PSP e de desenvolvimento de software. Esta análise pode ser feita para cada programa ao final da experiência, com valores médios e percentuais agrupando os dados de todos os programas. As análises apresentadas a seguir são semelhantes às descritas no item anterior, só que agora coletivamente.

A primeira análise realizada foi a análise de consistência dos dados. Foram verificados quais os tipos de erros cometidos pelos alunos ao preencher as planilhas do PSP. O Gráfico 1 expõe o percentual de ocorrência de cada tipo de erro. Os tipos de erros presentes no gráfico correspondem aos tipos descritos na Tabela 1. No que diz respeito à consistência dos dados, o gráfico mostra que os maiores problemas foram:

- A falta do planejamento do tempo e dos erros;
- Os alunos não registram todos os erros cometidos no formulário de log de erros (Defect Recording Log);
- A transferência dos dados dos formulários de logs de tempo e erros (Time Recording Log e Defect Recording Log) para o formulário de resumo (Project Plan Summary).

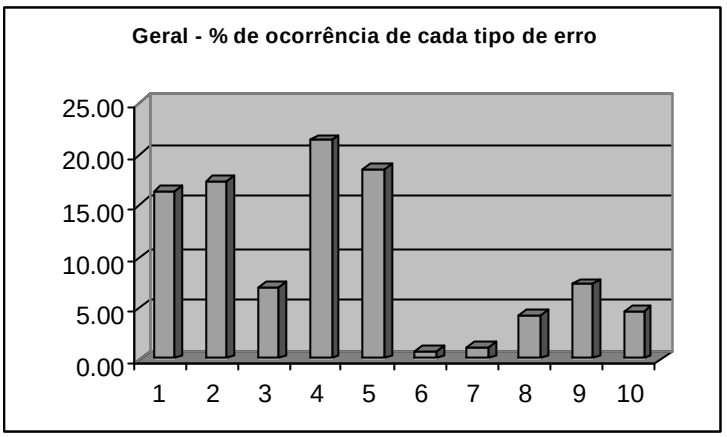

Gráfico 1: Análise de consistência dos dados

Analisando o Gráfico 2 com a distribuição de tempo nas fases do desenvolvimento do programa nota-se que a maior parte do tempo (42,29 %) foi dedicada a fase de codificação, o que era esperado pela natureza dos problemas propostos. Além disso, dois pontos chamam a atenção, um de forma positiva e outro de forma negativa. O ponto positivo foi que uma boa parcela do tempo foi gasto com análise e o projeto do programa (quase 20% somadas as duas fases). O ponto negativo foi a pequena parcela de tempo direcionada a revisão de código (5,67 %), o que mostra que essa ainda não é uma prática comum. As fases apresentadas neste e nos demais gráficos são: Análise (1), Projeto (2), Código (3), Revisão de código (4), Compilação (5), Testes (6), Postmortem (7).

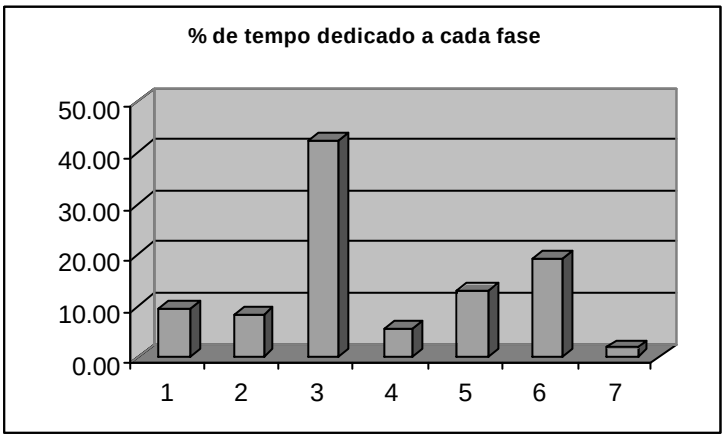

Gráfico 2: Percentual de tempo dedicado às fases

Com relação à análise das fases onde os erros foram cometidos, observou-se que a maior parte dos erros (quase 70%), foi cometida na fase de codificação, o que é esperado dada a natureza dos problemas propostos. Verifica-se que aproximadamente 10% dos erros foram inseridos na fase de análise ou projeto, o que é considerado negativo do ponto de vista de qualidade, pois são erros difíceis de serem detectados e que algumas vezes permanecem no programa sem se manifestarem.

Talvez o resultado mais preocupante em toda a análise foi o mostrado no Gráfico 3, onde foram registradas as fases onde os erros foram removidos. O que se pode notar é que quase a totalidade dos erros é removida nas fases de compilação e testes (mais de 80%). Em programas simples como o 1A, 2A e 3A, se a revisão de código fosse praticada de forma eficiente, praticamente todos os erros poderiam ser removidos nesta fase. Partindo-se da política que quanto mais tarde o erro é encontrado mais cara é a sua remoção, é essencial que o quadro apresentado seja revertido.

Com relação à comparação do tempo estimado e gasto em cada fase, a intenção não era verificar a capacidade de estimativa do aluno, mas somente mostrar a eles as dificuldades desta prática. De toda forma, o percentual médio de erro de estimativa foi alto, mais de 40%.

Enfim, o perfil da turma pode ser sumarizado na dedicação da maior parte do tempo a codificação, com pouca revisão de código. Isso acarreta muitos erros de codificação, que são removidos apenas nas fases de compilação ou testes, já que a revisão de código praticamente não é praticada.

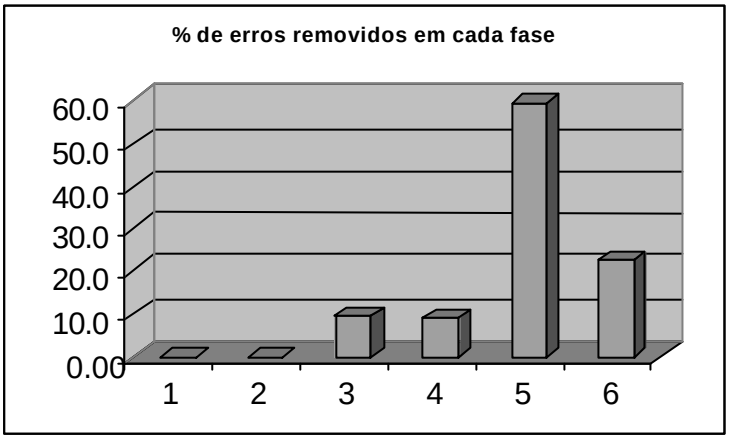

Gráfico 3: Percentual de erros removidos nas fases

Outras informações úteis podem ser colhidas, como por exemplo a verificação se a turma aumentou a dedicação à revisão de código no decorrer dos exercícios, ou se cometeu em média menos erros, ou ainda se conseguiu melhorar a eficiência na remoção dos erros. Para se ter uma análise bem feita, seria necessário avançar mais nos exercícios, seguindo toda a seqüência até o 10A. O tipo de análise realizada depende dos objetivos que se espera alcançar com a prática do PSP. Por exemplo, para este trabalho, procurou-se verificar a evolução da dedicação e eficiência da revisão de código para remoção de erros e foram desenvolvidos gráficos com esta finalidade.

5. Conclusões

A disciplina e o exercício da verdadeira Engenharia de Software deveriam ser colocados em prática o mais cedo quanto possível nos cursos de Computação e correlatos. O ensino e prática de técnicas como o PSP deveriam ser incluídas nos programas dos cursos e nos currículos de referência como conhecimento fundamental, pré-requisito até para as disciplinas que antecedem às de Engenharia de Software.

Para aplicação completa dos exercícios do PSP seria necessário uma disciplina exclusivamente para este propósito. Certamente os resultados seriam ainda melhores que os obtidos com a aplicação de uma parte do curso. No entanto, não havendo esta possibilidade, a aplicação de um subconjunto dos exercícios é uma opção viável, não compromete o curso normal da disciplina e com certeza já traz benefícios significativos para os alunos. No caso desta experiência com o PSP, não há dados suficientes para comprovar os resultados positivos, mas melhorias na percepção da necessidade e dos conceitos de qualidade podem ser percebidas no contato e depoimentos dos alunos envolvidos, na motivação destes, etc.

Para futuras aplicações do curso de PSP seria importante, antes da aplicação dos exercícios, chamar a atenção dos alunos para os erros de preenchimento mais comumente cometidos (Tabela 1), para que dados não sejam perdidos. Visando uma melhoria continua da prática de desenvolvimento de software dos alunos, é fundamental também, que após a aplicação de cada exercício e antes da aplicação do exercício seguinte os resultados das análises de consistência e qualidade sejam repassados aos alunos. Desta forma, os estudantes ficariam conscientes dos erros cometidos no exercício anterior para não cometê-los novamente. Nas primeiras experiências na disciplina de Engenharia de Software, a exposição dos resultados para a turma foi realizada somente após a aplicação de todos os exercícios. Já na experiência relatada neste trabalho foi dado o *feedback* para os alunos após a aplicação de cada exercício, o que permitiu alcançar resultados muito melhores.

6. Referências Bibliográficas

Dion and Raymond "Process Improvement and the Corporate Balance Sheet", IEEE Software, July 1993.

Hilburn, T., Hirmanpour, I. and Kornecki, A. "The Integration of Software Engineering into a Computer Science Curriculum. Lecture Notes in Computer Science: Software Engineering Education", Proceedings of the Eight SEI Conference on Software Engineering Education, 1995.

Humphrey, W. S. "A Discipline for Software Engineering", Carnegie Mellon University: Addison-Wesley Publishing Company, 1995.

Humphrey, W. S. "Using a Defined and Measured Personal Software Process", IEEE Software 13(3), 77-88, 1996.

Pressman, R. S. "Engenharia de Software", 4 ed. São Paulo: Editora MAKRON Books do Brasil Editora Ltda., 2002.

Towhidnejad, M., Hilburn, T. "Integrating the Personal Software Process (PSP) across the Undergraduate Curriculum", Proc. 27th Frontiers in Education Conference, 1997.