Detectando diferenças significativas entre programas como auxílio ao aprendizado colaborativo de programação

Eustáquio São José de Faria [1], Juan Manuel Adán Coello[2]

[1] Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais – Arcos, MG – Brasil

[2]Faculdade de Engenharia de Computação
Pontifícia Universidade Católica de Campinas – Campinas, SP, Brasil

eustáquio@pucminas.br, juan@puc-campinas.edu.br

Abstract. *The analysis of computer programs to find significative differences in student programs is proposed to support collaborative learning of programming skills, following Neo-Piagetian ideas of the socio-cognitive conflict theory. Several metrics can be used to find the level of experience and the programmer's ability in solving problems in a pre-defined language. We suggested a comparison among student programs in order to find significative differences among them. Once analyzed, student programs can be distributed among classmates whose programs have significative differences, so that they can be discussed and serve as example and counterexample of good and bad programming practices. In this article we discuss some programs quality metrics and propose a tool called VDSP (*Verificador de Diferenças Significativas em Programas*) that automatically computes those metrics and identify significative differences for C programs.*

Keywords. *Significant difference between programss, collaborative learning of programming, computer program quality, software metrics.*

Resumo. *A idéia de analisar programas de computador construídos por alunos para encontrar diferenças significativas surge como apoio ao aprendizado colaborativo de programação, seguindo as idéias neopiagetianas da teoria do conflito sócio-cognitivo. Para tanto, faz-se necessária uma análise dos programas para computar métricas que possam indicar o nível de experiência e habilidade do programador em resolver problemas em uma linguagem pré-definida. Sugerimos uma comparação entre os programas analisados a fim de detectar diferenças significativas entre eles. Os autores de programas que apresentam diferenças significativas podem então trocar os seus programas para que sejam apreciados e sirvam de exemplo e contra-exemplo de boas e más práticas de programação. Neste artigo vamos apresentar algumas métricas de qualidade de programas e propor a construção de um avaliador para programas escritos em linguagem C, chamado VDSP – Verificador de Diferenças Significativas em Programas.*

Palavras-chave. *Diferenças significativas de programas, aprendizado colaborativo de programação, qualidade de programa de computador, métricas de software.*

1. Introdução

A literatura e a nossa vivência profissional mostram que os alunos iniciantes em cursos de computação enfrentam vários problemas relacionados com o desenvolvimento de habilidades de programação de computadores, principalmente em currículos onde se inicia o curso com o estudo aprofundado de programação.

Tobar et al. (2001) argumentam que vários podem ser os motivos para a experiência frustrante dos alunos na construção dos seus primeiros programas: o excesso de preocupação com detalhes sintáticos e semânticos da linguagem usada; a falta de uma visão daquilo que se quer solucionar, de idealizar soluções adequadas, de mapear essas soluções em passos seqüenciais e de abstrair o funcionamento dos mecanismos escolhidos.

Proulx (2000) destaca dois motivos para os problemas com programação que os novatos enfrentam, são eles: (1) a complexidade crescente dos paradigmas de programação; (2) o desconhecimento de padrões de raciocínio lógico na resolução de problemas.

Outros motivos que consideramos importantes são: a falta de prática na solução de problemas computacionais; a concepção já internalizada nos alunos de que a solução de problemas nada mais é que aplicação de fórmulas já estabelecidas; a falta de costume com a atividade de criação; a falta de incentivo dos professores das disciplinas de programação para o trabalho colaborativo; etc.

Analisando a bibliografia e diversas pesquisas na área, percebemos que o problema é bastante antigo e, para tentar resolvê-lo, sugerimos o uso de tutores automatizados para ensino de programação combinado com várias estratégias de ensino: estratégia de conclusão, de trabalho em dupla, estratégia construtivista de ensino e estratégia de colaboração. Neste artigo, vamos focar a estratégia de colaboração.

2. Estratégia de Colaboração

Tobar et. al (2001) apontam o aprendizado colaborativo como uma boa metodologia para sanar, ou pelo menos amenizar, este problema há muito tempo notado em instituições de ensino médio e superior que ministram cursos de informática.

A colaboração parte do princípio de que dois ou mais indivíduos, trabalhando conjuntamente, podem chegar a uma situação de equilíbrio, onde as idéias possam ser trocadas, distribuídas entre os participantes do grupo, gerando assim, novas idéias e conhecimentos, frutos deste trabalho coletivo (LÉVY, 1999).

Segundo Larocque (apud SANTAROSA, 1999), grande parte das pessoas, ao ser questionada sobre recordações de experiências envolvendo uma situação de aprendizagem, lembra-se de alguma situação envolvendo outras pessoas. Apesar disto, a maioria das metodologias pedagógicas e os métodos envolvendo novas tecnologias, privilegiam situações ou contextos de aprendizagem individual. Mesmo assim, Santarosa (1999) aponta para o crescimento da quantidade de pesquisas envolvendo a promoção da aprendizagem utilizando as vantagens do convívio social e a aprendizagem colaborativa.

A nossa proposta é construir um software que detecte as diferenças significativas nos programas construídos por alunos de cursos de informática, com o intuito de incentivar a formação de equipes para aprendizado colaborativo, baseado na teoria Neo-Piagetiana do conflito sócio-cognitivo, que defende a importância das discussões e conflitos entre as idéias dos alunos no processo de aprendizagem.

Para trabalharmos com a idéia do conflito sócio-cognitivo, no contexto do aprendizado colaborativo de programação, precisamos identificar diferenças significativas nos programas desenvolvidos pelos alunos. Essas diferenças devem ser importantes o bastante para encorajar os alunos a discuti-las, ou seja, devem ter um significado tal que faça com que os alunos envolvidos na discussão tenham um ganho de conhecimento advindo do conflito.

Nesta pesquisa, o termo "diferenças significativas" está relacionado a algumas métricas de qualidade de programas, que, de uma certa forma, caracterizam a lógica ou o estilo de programação utilizado pelo autor.

Independente do nível de conhecimento dos participantes de grupos colaborativos é interessante uma troca de programas entre estes para uma análise e reflexão do trabalho executado pelos colegas.

3. Qualidade de Programas

A qualidade de um programa de computador está diretamente ligada ao uso de técnicas de organização estrutural e lógica em sua construção. Não compreendemos como programas de qualidade aqueles que simplesmente retornam um resultado correto. Além desta característica essencial, devemos analisar o uso das estruturas lógicas de controle, a documentação (comentários), a disposição das linhas de comando (endentação), o respeito ao acoplamento, a modularização das funções do programa, etc.

Para Sommerville (2003), da mesma maneira que os serviços que os programas fornecem, eles têm uma série de outros atributos associados, que refletem sua qualidade. Esses atributos não são diretamente associados ao que o programa faz. Em vez disso, eles refletem seu comportamento quando em funcionamento, a estrutura e a organização do código fonte e também a documentação associada. Exemplos desses atributos (às vezes, chamados de atributos funcionais) são o tempo de detecção e resolução de um problema de lógica no programa e a facilidade de compreensão do código do programa.

Segundo Toscani e Veloso (2001), muitos destes atributos, relacionados à qualidade de programas, são empiricamente conhecidos dos programadores experientes, mas complicados de passar para um iniciante, por exigirem muita abstração e uso prático, por não estarem todos fundamentados, etc.

Muitos estudos sobre métricas de qualidade de *software* foram feitos nos últimos anos, dentre eles podemos citar Berry e Meekings (1985) e Hung e Chang (1993), que construíram avaliadores automatizados de programas baseados em algumas métricas levantadas por eles; Schorsch (1995), que deu origem a uma ferramenta automatizada de avaliação de programas em Pascal baseada em erros de lógica e sintaxe e em métricas de estilo; Mengel e Yerramilli (1999) que destacaram a utilidade de métricas para medir a qualidade dos programas dos alunos implementadas em um avaliador automatizado comercial chamado Verilog Logiscope; Xenos et. al (2000) que estudaram o uso de

métricas para medir a qualidade de programas orientados a objetos; Jackson (2000) que descreve uma abordagem semi-automatizada para avaliação on-line, levantando métricas de qualidade que devem ser consideradas para avaliação de programas; Purao e Vaishnavi (2003), que fazem uma avaliação de várias métricas de qualidade de programas orientados a objetos.

4. Métricas de Software

O maior desafio é definir claramente o que podem ser consideradas diferenças significativas, ou seja, quais métricas poderão ser usadas para definir as diferenças significativas entre os programas. O que devemos considerar como diferença significativa entre dois programas, a ponto de ajudar no aprendizado colaborativo, se foram construídos para resolver o mesmo problema?

Jackson (1996), baseado no estudo de Hung, Kwork e Chan (1993), discute sobre o que podem ser possíveis métricas para analisar programas de estudantes de um modo automatizado. Segundo ele, podem ser usados os seguintes critérios:

- Corretude – tem a ver com a produção e estrutura do programa conforme as exigências colocadas pelo professor.
- Estilo de escrita – tem a ver com o comprimento de módulo, comprimento de identificador, linhas de comentário, endentação, linhas em branco, uso de constantes e outros aspectos relacionados ao estilo de construção do programa.
- Eficiência – tem a ver com o tempo de CPU usado pelo programa do estudante comparado ao programa do professor.
- Complexidade – tem a ver com a métrica de complexidade ciclomática de McCabe.

Cada um dos critérios mostrados acima pode ser avaliado com uma gama extensa de métricas.

Oman e Cook (1990), em um estudo sobre taxonomia para estilo de programação, argumentaram que estilo de programação é pouco ensinado em cursos de computação e é discutido em pequena escala na maioria dos livros de programação. Em geral, pouca ênfase é dada à importância do uso de um bom estilo para garantia de boa programação. O surpreendente é que já se passou mais de uma década desde a publicação do trabalho de Oman e Cook, e pouco foi feito para reverter este quadro. Isso nos incentivou a escolher métricas baseadas em estilo para implementação do VDSP.

Nosso maior interesse é verificar se o VDSP contribuirá na elaboração de grupos colaborativos e se o trabalho colaborativo pode realmente trazer melhorias para o aprendizado de programação. Não é nossa intenção levantar todas as métricas para construção do avaliador automatizado. Fica para uma pesquisa futura um estudo aprofundado de quais métricas seriam ideais, de modo a atingir todos os critérios definidos por Jackson (1996).

Para a construção do VDSP, foram escolhidas as métricas de estilo de escrita mais encontradas na literatura. Essas métricas são fáceis de implementar em um avaliador automatizado e medem a capacidade de um programador em escrever código de maneira organizada, são elas:

1. Tamanho de identificadores;
2. Percentual de identificadores que são constantes (define / const);
3. Tamanho dos módulos;
4. Quantidade de módulos;
5. Percentual de linhas endentadas;
6. Percentual de linhas de comentário;
7. Percentual de linhas em branco.

5. Verificador de Diferenças Significativas em Programas - VDSP

O VDSP é um avaliador automatizado de programas em linguagem C que deverá ser usado como auxílio à montagem de grupos colaborativos de aprendizado de programação, e fornecerá um relatório aos alunos das anomalias de estilo encontradas nos programas avaliados.

Todo programa avaliado receberá um valor correspondente a um índice global relativo à qualidade do estilo de programação usada em sua construção.

5.1. Funcionamento geral do VDSP

O avaliador automatizado de programas (VDSP) deverá percorrer o programa a ser avaliado para atribuir valores a cada métrica mostrada na seção 4 deste artigo.

Seguindo o exemplo de Berry e Meekings (1985) e Hung e Chan (1993), toda atividade a ser avaliada deve estar relacionada a um conjunto mestre de valores aceitáveis máximos e mínimos para cada métrica, definidos pelo instrutor. Dentro da faixa de valores máximos e mínimos aceitáveis teremos um conjunto de valores ideais máximos e mínimos também definidos pelo instrutor. Por exemplo, para a tarefa 1, os valores máximo e mínimo aceitáveis para a métrica **percentual de linhas endentadas** deverão ser armazenados. Dentro deste limite armazenaremos também os valores máximo e mínimo ideais para a mesma métrica. Assim, todo programa entregue pelos alunos que corresponder à tarefa 1 deverá ser avaliado de acordo com os valores máximos e mínimos aceitáveis e ideais para esta tarefa. Qualquer valor dentro daquela gama de valores ideais receberá o maior número de pontos possível, que será estipulado em 100 pontos para cada métrica, perfazendo um total de 700 pontos, uma vez que definimos o uso de sete métricas para avaliação dos programas. Qualquer valor entre o máximo e mínimo, mas fora da gama ideal, receberá pontos baseados no quão distante da gama ideal ele está (valor proporcional métrico). Os valores fora do limite mínimo e máximo aceitável não receberão pontuação. O valor final computado, correspondente à soma de todos valores atribuídos às métricas analisadas no programa, será o índice global que representa a qualidade do programa.

Um relatório de anomalias, em relação às métricas, deverá ser gerado e disponibilizado ao autor do programa avaliado, juntamente com o índice global de qualidade computado.

5.1.1. Cálculo do Valor Proporcional Métrico

Como dito na seção 5.1, um valor proporcional deverá ser calculado para cada métrica quando o valor encontrado pelo avaliador automatizado para um dado programa estiver dentro da gama aceitável e fora da gama ideal de valores para aquela métrica. Consideremos Valor Mínimo Aceitável como VMinA, Valor Mínimo Ideal como VMinI, Valor Máximo Ideal como VMaxI, Valor Máximo Aceitável como VMaxA, Valor Encontrado pelo avaliador como VE e Valor Atribuído à métrica como VA. O cálculo deverá ser feito da seguinte maneira:

Quando o valor da métrica encontrado no programa avaliado estiver acima do mínimo aceitável e abaixo do mínimo ideal:

VA = ((VE – VMinA) / (VMinI – VMinA))*100

Quando o valor da métrica encontrado no programa avaliado estiver acima do máximo ideal e abaixo do máximo aceitável:

VA = ((VMaxA – VE) / (VMaxA – VMaxI))*100

Exemplo 1:

Após análise do VDSP foi encontrado um valor 6 para a métrica 4 (quantidade de módulos). Esta métrica recebeu VMinA = 5, VMinI = 7, VMaxI = 8 e VMaxA = 12 para esta tarefa. Notemos que o valor 6 está acima de VMinA e abaixo de VMinI, portanto usamos a primeira fórmula descrita acima.

VA = ((VE – VMinA) / (VMinI – VMinA))*100

VA = ((6 – 5) / (7 – 5))*100

VA = ((1) / (2))*100

VA = 50

VA = 50 é o valor que deverá ser atribuído para a métrica 4 do programa que está sendo avaliado, e deverá ser adicionado ao índice global de qualidade do programa.

Exemplo 2:

Após análise do VDSP foi encontrado um valor 22 para a métrica 6 (percentual de linhas de comentário). Esta métrica recebeu VMinA = 10, VMinI = 15, VMaxI = 20 e VMaxA = 30 para esta tarefa. Notemos que o valor 22 está acima de VMaxI e abaixo de VMaxA, portanto usamos a segunda fórmula.

VA = ((VMaxA – VE) / (VMaxA – VMaxI))*100

VA = ((30 – 22) / (30 – 20))*100

VA = ((8) / (10))*100

VA = 80

VA = 80 é o valor que deverá ser atribuído para a métrica 6 do programa que está sendo avaliado, e também deverá ser adicionado ao índice global de qualidade do programa.

O cálculo proporcional métrico será calculado, se necessário, uma única vez para todas as métricas, exceto para as métricas **tamanho de identificadores** e **tamanho dos módulos**. Calcularemos o valor proporcional métrico para cada identificador e, ao final, uma média dos valores proporcionais métricos para os identificadores corresponderá ao valor atribuído pelo VDSP para esta métrica. O mesmo acontecerá com a métrica tamanho dos módulos. Os valores atribuídos considerados ideais para estas duas métricas devem ser superiores a 80.

5.1.2. Relatório de Anomalias

Um relatório deverá ser disponibilizado para o autor do programa contendo todos os valores computados para cada métrica juntamente com seu valor atribuído pelo VDSP e anomalias e soluções.Vejamos um exemplo:

O programa do aluno **José** é avaliado pelo VDSP e detecta-se que a métrica tamanho dos identificadores recebeu pontuação 40, isto significa dizer que a média de pontuação atribuída para os identificadores é 40, que corresponde a um valor baixo. Podemos considerar esta não conformidade como uma anomalia. O percentual de identificadores que são constantes (define/const) é de 2% enquanto o valor mínimo aceitável está estipulado pelo instrutor é de 5%. Mais métricas são analisadas e detecta-se que a métrica tamanho dos módulos recebeu pontuação 92, que corresponde a um ótimo valor. Para a métrica quantidade de módulos é encontrado o valor 4, enquanto o valor mínimo aceitável para esta métrica é de 3 e o mínimo ideal é de 5, assim, não consideramos uma anomalia, mas uma atenção deve ser dada a esta métrica. As demais métricas analisadas se encontram dentro do limite ideal para elas.

Ao final da avaliação um relatório é gerado, como abaixo, apontando os seguintes resultados:

Métrica 1.

Tamanho encontrado de identificadores → 40

Valor atribuído (VA1) → 40

Anomalia: O programa avaliado possui muitos identificadores com tamanho abaixo do tamanho ideal.

Solução: Elaborar nomes mais sugestivos para os identificadores (nomes de variáveis, nomes de funções e procedimentos, nomes de constantes, nomes de tipos novos).

Métrica 2.

Percentual encontrado de identificadores que são constantes (define/const) → 2%

Valor atribuído (VA2) → 0

Anomalia: O programa avaliado possui poucos identificadores que são constantes definidas.

Solução: Existem situações no programa onde o uso de constantes aumentaria a manutenibilidade do programa. Descubra estas situações e procure substituir os valores utilizados por constantes.

Métrica 3.

Tamanho encontrado dos módulos → 92

Valor atribuído (VA3) → 92

Anomalia: Não existe.

Solução: Não existe.

Métrica 4.

Quantidade encontrada de módulos → 4

Valor atribuído (VA4) → 50

Anomalia: Não existe, mas atenção para a quantidade de módulos do programa.

Solução: Procure descobrir situações onde algum módulo possa ser subdividido em mais módulos.

As demais métricas serão mostradas com seus respectivos valores, que corresponderão a 100 (por estarem dentro do limite ideal), e com a indicação de que não há anomalias. O índice global de qualidade será mostrado ao final do relatório.

Os valores mínimos e máximos (ideais e aceitáveis) para cada métrica, armazenados no VDSP pelo instrutor, não serão mostrados aos alunos. A intenção é fazer com que os alunos discutam suas anomalias e procurem detectar, com o processo do conflito sócio cognitivo, quais valores seriam ideais e como solucionar o problema.

Este relatório deverá ser impresso pelo aluno José para uso durante o trabalho colaborativo em grupo.

5.1.3. Índice Global de Qualidade do Programa (IG)

Como dito anteriormente, o valor do índice global de qualidade do programa avaliado será calculado pela soma de todos os valores atribuídos pelo VDSP às métricas.

Usando ainda o exemplo acima, vamos calcular o índice global:

$IG = VA_1 + VA_2 + VA_3 + VA_4 + VA_5 + VA_6 + VA_7$

$IG = 40 + 0 + 92 + 50 + 100 + 100 + 100$

$IG = 482$

6. Experimentação com o VDSP

A experimentação será feita com a turma do 2º período de Sistemas de Informação da PUC Minas Arcos, cursando a disciplina Algoritmos e Estruturas de Dados – AED, que possui 35 alunos. Esta turma foi escolhida porque é uma turma iniciante, com pouca experiência, mas com conhecimentos suficientes para solucionar os problemas propostos.

Serão realizadas quatro tarefas de programação. Inicialmente, a primeira será realizada de maneira individual e, após a avaliação desta tarefa pelo VDSP, grupos colaborativos serão montados para construção da tarefa novamente. A segunda e terceira

tarefas seguirão o mesmo ciclo da primeira e a quarta tarefa será realizada individualmente, sendo avaliada somente uma vez.

A primeira tarefa será aquela em que faremos o levantamento do nível da turma e cuja pontuação, atribuída pelo VDSP, será utilizada para montagem dos grupos. Seis grupos serão formados por quatro componentes e terão composição homogênea e heterogênea de pontuação. Os outros onze alunos realizarão as tarefas individualmente. A intenção desta estratégia é medir o ganho de aprendizado advindo da colaboração em grupos homogêneos relativamente a grupos heterogêneos, em grupos homogêneos relativamente a alunos que não formaram grupos e em grupos heterogêneos relativamente a alunos que não formaram grupos.

A colaboração acontecerá em sala de aula prática da disciplina, no laboratório de programação. Os alunos deverão estar de posse dos seus programas individuais bem como do relatório gerado pelo VDSP. Será solicitado que os alunos de um mesmo grupo troquem os relatórios e os programas e procurem encontrar as anomalias relatadas no programa do colega. Todo aluno irá expor as possíveis fontes de anomalias encontradas no programa que está analisando. Um líder de grupo, preparado pelo professor para intermediar a colaboração entre os colegas, conduzirá a construção de um novo programa. Este novo programa será avaliado pelo VDSP e um novo relatório será gerado para os alunos e o instrutor verificarem o ganho de conhecimento adquirido pelo grupo. Os alunos serão solicitados a comentarem as diferenças entre o seu programa e o do grupo naqueles aspectos em que há diferenças importantes. Paralelamente, os alunos que fizerem a tarefa individual também terão o seu novo programa submetido ao avaliador automatizado. Este ciclo se repetirá para as tarefas 2 e 3.

Uma quarta tarefa, de nível de complexidade semelhante, será solicitada aos alunos. Esta tarefa será executada individualmente. Cada aluno deverá submeter o seu programa ao avaliador automatizado.

A primeira, a segunda e a terceira tarefas serão usadas pelo instrutor para medir o ganho de aprendizado do grupo advindo da colaboração.

A quarta tarefa será usada pelo instrutor para medir o ganho de aprendizado individual advindo da colaboração.

Todos os programas avaliados serão armazenados, bem como sua avaliação, para serem comparados com os programas que serão entregues pelos grupos.

7. Trabalho Similar

Um trabalho semelhante foi desenvolvido por Constantino-Gonzalez e Suthers (2000) para o aprendizado colaborativo de modelagem entidade-relacionamento. Seguindo também as idéias da teoria do conflito sócio-cognitivo, os autores desenvolveram um ambiente para aprendizado colaborativo de modelagem entidade-relacionamento mediado por computador (COLER), onde os alunos podem desenvolver suas tarefas na presença de outros colegas, ou individualmente, se assim o desejarem, e mais adiante desenvolver a mesma tarefa em grupos colaborativos. Os diagramas são submetidos a avaliação e depois são comparados com os dos colegas para encontrar diferenças significativas. São formados pequenos grupos de discussão mediados pelo COLER e um novo diagrama é construído pelo grupo. Durante o período de mediação do trabalho em

grupo o COLER interfere na construção do novo diagrama, dando sugestões baseadas nos diagramas construídos individualmente e nos grupos de outros colegas, monitorando a participação de todos os componentes, incentivando aqueles que não estão trabalhando o suficiente e alertando aqueles que trabalham em excesso.

Da mesma maneira que o COLER, incentivaremos o trabalho colaborativo baseados na teoria Neo-Piagetiana do conflito sócio cognitivo. Entretanto, a mediação executada pelo COLER para garantir a colaboração, em nosso projeto, será feita manualmente, em aulas práticas de programação.

4. Referências

BERRY, R.E. and MEEKINGS, B.A.E. (1985) "A Style Analysis of C Programs", communications of the ACM, volume 28, issue 1, january, p. 80-88.

CONSTANTINO-GONZÁLEZ, M.A. and SUTHERS, D.D. (2000) "A Coached Collaborative Learning Environment for Entity-Relationship Modeling", In: Intelligent Tutoring Systems, proceedings of the 5th International Conference (ITS 2000), G. Gauthier, C. Frasson and K. Van Lehn (Eds.), Springer-Verlag, p. 325-333.

HUNG, S., KWORK, L., and CHAN, R. (1993) "Automatic Program Assessment", Computers and Education, volume 20, issue 2, p. 183-190.

JACKSON, David. (1996) "A Software System for Grading Student Computer Programs", Computers and Education, vol. 27, issue 314, p. 171–180.

JACKSON, David. (2000) "A Semi-Automated Approach to Online Assessment", iTiCSE 2000, Helsinki, Finland, july, p. 164-167.

LÉVY, Pierre. A Inteligência Coletiva, 2ª ed, Loyola, 1999.

MENGEL, Susan A. and YERRAMILLI, Vinay. (1999) "A Case Study Of The Static Analysis Of The Quality Of Novice Student Programs", SIGCSE'99, New Orleans, LA, USA, march, p. 78-82.

OMAN, Paul W. and COOK, Curtis R. (1990) "A Taxonomy For Programming Style", ACM, may, p. 244-250.

PROULX, V.K. (2000) "Programming Patterns and Design Patterns in the Introductory Computer Science Course", proc. of the 31st SIGCSE, Austin, TX, USA, march, p. 7-12.

PURAO, Sandeep and VAISHNAVI, Vijay. (2003) "Product Metrics for Object-Oriented Systems", ACM Computing Surveys, vol. 35, number 2, june, p. 191-221.

SANTAROSA, Lucila Costi. (1999) "Criação de Ambientes de Aprendizagem Colaborativa", SBIE – X, Curitiba/PR, novembro.

SCHORSCH, Tom. (1995) "Cap: An Automated Self-Assessment Tool To Check Pascal Programs For Syntax, Logic And Style Errors", SIGCSE'95, Nashville, TN, USA, march, p. 168-172.

SOMMERVILLE, Ian. "Engenharia de Software". 6ª ed., Addison Wesley, 2003.

TOBAR, Carlos Miguel, GARCIA ROSA, João Luís, ADÁN COELLO, Juan Manuel and PANNAIN, Ricardo. (2001) "Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação", SBIE – XII, Vitória/ES, novembro.

TOSCANI, Laira Vieira and VELOSO, Paulo A. S. (2001) "Complexidade de Algoritmos – Análise, Projeto e Métodos", número 13, Sagra Luzzatto, 2001.

XENOS, M., STAVRINOUDIS, D., ZIKOULI, K. & CHRISTODOULAKIS, D. (2000) "Object-oriented metrics – a survey", proc. of the FESMA - Federation of European Software Measurement Associations, Madrid, Spain, p. 1-10.