Aprendizagem de iniciantes em algoritmos e programação: foco nas competências de autoavaliação

Silvério Sirotheau[1], Silvana Rossy de Brito[2], Aleksandra do Socorro da Silva[2], Marianne Kogut Eliasquevici[1], Eloi Luiz Favero[1], Orivaldo de Lira Tavares[3]

[1]Universidade Federal do Pará (UFPA)
Caixa Postal 479 – 66.075-110 – Belém – Pará – Brasil

[2] Universidade Federal Rural da Amazônia (UFRA)
Caixa Postal 917 – 66.077-530 – Belém – Pará – Brasil

[3]Universidade Federal do Espírito Santo (UFES)
29.060-900 – Vitória – ES – Brasil

{ssirotheau@gmail.com, silvana.rossy@ufra.edu.br, aleksandra.silva@ufra.edu.br, mariufpa@gmail.com, favero@ufpa.br, tavares@inf.ufes.br}

Abstract.

This article demonstrates the incorporation in JavaTool, already in use and integrated into the Moodle platform, features that enable the exercise of the feedback from students as a way to stimulate the development of evaluation skills. Research has been carried out in order to demonstrate the skills of self-assessment focusing efforts on the proposed collaborative feedback subsystem. This article brings an important contribution to the discussion of automatic evaluation not only as an assessment tool, in order to reduce the workload of the teacher, but as a component that supports the processes of self-regulation of learning.

Resumo.

Este artigo demonstra a incorporação no JavaTool, já em uso e integrado à plataforma Moodle, recursos que possibilitem o exercício do feedback entre estudantes, como forma de estimular o desenvolvimento de habilidades de avaliação. Pesquisas têm sido realizadas com o intuito demostrar a competências de autoavaliação concentrando-se esforços na proposta do subsistema de feedback colaborativo. Este artigo traz uma contribuição importante na discussão da avaliação automática não apenas como um instrumento de avaliação, com a finalidade de reduzir a sobrecarga do professor, mas como um componente que apóia os processos de autorregulação da aprendizagem.

1. Introdução

A primeira experiência no aprendizado de programação para muitos estudantes da área de computação costuma ser frustrante. Dentre os motivos para esta frustração destacam-se: a preocupação excessiva com detalhes de sintaxe da linguagem sendo usada; a falta de uma visão daquilo que se quer solucionar, de idealizar soluções adequadas, de

mapear essas soluções em passos sequenciais e de abstrair o funcionamento dos mecanismos escolhidos; o estabelecimento de um raciocínio lógico visando à resolução de problemas, com base em um modelo incremental, em relação à complexidade e à estratégia de refinamentos sucessivos (Proulx 2000).

O esforço empreendido na construção de algoritmos e programas tende a se transformar em obstáculos que resultam em situações problemáticas, tais como: alto índice de repetência; acúmulo de dificuldades que influenciam nos níveis de evasão nos cursos; além de dificuldades demonstradas nas disciplinas diretamente dependentes das habilidades de programar, dominar o raciocínio lógico e resolver problemas.

Para minimizar essas dificuldades, professores dedicam esforços no ensino dos aspectos sintáticos e semânticos das linguagens de programação. A abordagem didática, com frequência envolve a apresentação de exemplos com programas de estruturas semelhantes àquelas a serem aplicadas nesses primeiros exercícios de programação. Segundo Menezes *et*. al. (2008), essa estratégia está baseada no fato de que os estudantes precisam criar, em suas memórias, padrões de estruturas que possam ser usadas para resolver diferentes tipos de problemas. Outra possibilidade de auxiliar os estudantes é o desenvolvimento de competências de autoavaliação por parte destes como forma de apoiá-los na compreensão dos objetivos da disciplina, no seu desempenho e nas suas estratégias utilizadas para reduzir ou eliminar a lacuna entre o desempenho alcançado e o esperado (Sadler 1998).

Diante deste contexto, este artigo expõe a incorporação em um ambiente de aprendizagem de programação, já em uso e integrado à plataforma Moodle (Mota et al. 2009), de recursos que possibilitem o exercício do *feedback* entre estudantes, como forma de estimular o desenvolvimento de habilidades de avaliação nos estudantes. A proposta está baseada na especificação de novos requisitos, realizada a partir da avaliação de utilização do sistema pelos estudantes e de uma nova revisão na literatura em torno dos problemas no ensino-aprendizagem de programação.

O artigo está organizado da seguinte forma: a seção 2 apresenta a importância do *feedback* colaborativo na autorregulação da aprendizagem de programação e alguns trabalhos correlatados; a seção 3 descreve a evolução da pesquisa e os avanços em direção à autorregulação da aprendizagem; a seção 4 discorre sobre a Discussões da autoavaliação e, a última seção tece as considerações finais sobre o trabalho desenvolvido e direções para pesquisas futuras.

2. A importância do *feedback* colaborativo na autorregulação da aprendizagem de programação e trabalhos correlatados

Segundo Rust et al. (2003), existem várias pesquisas que comprovam que estudantes com desempenho menor do que o esperado possuem baixa compreensão quanto aos requisitos das tarefas que lhes foram propostas. Butler e Winne (1995) revelaram que a eficácia dos estudantes está relacionada com a capacidade de usar as informações recebidas, a partir do acompanhamento dos professores, para alcançar os objetivos esperados. O feedback é considerado um importante elemento para aprendizagem, visto que permite ressaltar as dissonâncias entre o resultado pretendido e o real. Entretanto, quando pouco eficaz ou demorado prejudica a percepção do estudante quanto ao seu progresso, o que pode reduzir sua motivação (Schunk, 1989).

A aprendizagem centrada no estudante é descrita como um processo no qual ele constrói ativamente seu próprio conhecimento e habilidades (Barr e Tagg 1995). Desta forma, o estudante interage com o conteúdo transformando e discutindo o assunto, a fim de internalizar os significados e fazer conexões com o que ele já sabe. Conforme Lea et al. (2003), os pressupostos fundamentais são o engajamento ativo e a responsabilidade do estudante pela gestão da aprendizagem. Algumas práticas podem tornar a aprendizagem mais ativa (Shang et al. 2001): ampliar as experiências com a criação de pequenos grupos, colocando-os em situações de tomadas de decisão ou sugerindo questões; tirar vantagem do "poder de interação", de forma que as várias atividades de aprendizagem possam multiplicar o impacto educacional.

De acordo com Santos (2002), a coavaliação entre pares é um processo de regulação que oferece potencialidades e reconhece a interação social como um recurso fundamental na construção do conhecimento, visto que os alunos são colocados em situações de confronto, de troca e de decisão, e também são forçados a argumentar, expor ideias, planejar, dividir o trabalho, entre outras circunstâncias. No ensino e aprendizagem de soluções de programação, por exemplo, após a construção de uma solução computacional, os estudantes podem ser convidados a escrever sobre o trajeto realizado para chegar à solução e discutir os resultados e as decisões adotadas com seus pares, como realizado por Menezes et al. (2008), em uma abordagem adaptada de Polya (1978). Essa oportunidade, de relacionar a experiência com o diálogo, oferece aos estudantes uma perspectiva nova sobre as suas convicções e valores, ajudando-os a construir os muitos possíveis significados para as suas experiências (Shang et al. 2001).

Esse cenário dá espaço para que o feedback não seja apenas produzido pelos professores ou pelas tecnologias computacionais, com conteúdo sobre o que está correto ou errado, seus pontos fortes e fracos, não permitindo que os estudantes formem suas habilidades de autoavaliação, as quais favorecem o desenvolvimento da autonomia dos mesmos.

Na aprendizagem de algoritmos e programação, grande parte dos trabalhos que expandem a concepção de feedback como "transmissão de informações" pelo professor, exploram a programação em pares, as soluções cooperativas e colaborativas entre estudantes com diferentes níveis de conhecimento e interesses, potencializando os efeitos das atividades individuais.

3. Evolução da pesquisa e os avanços em direção à autorregulação da aprendizagem

Objetivando contribuir para a compreensão do estudante no aprendizado de conceitos introdutórios de programação, foi implementado um visualizador e simulador de programas denominado JavaTool (Mota et al. 2008), inspirado na ideia do JEliot (Moreno *et al*. 2004), embora com representação gráfica diferente. Nas concepções iniciais da pesquisa, o foco da investigação estava nos benefícios da visualização dos programas. Dentre as categorias de visualizadores de programas, o BALSA (Brown 1988) destaca-se como um dos pioneiros e o como uma das tecnologias atuais em evidência, pois possibilita visualizar e animar código em Java, inclusive mostrando as estruturas de dados. No JavaTool o estudante edita, compila, depura e visualiza um código Java dinamicamente, da execução a sua solução, sendo mais indicado para o ensino de programadores iniciantes, pois não inclui, por exemplo, conceitos avançados

da orientação a objetos. Mota et al. (2009) incorporou o simulador e visualizador de programas à plataforma Moodle (2010), o que permitiu inserir recursos e atividades, além de favorecer o uso de estratégias didáticas que valorizam a interatividade por meio das ferramentas de comunicação já disponíveis na plataforma.

Para viabilizar a avaliação automática, foi incorporado por Moreira e Favero (2009) um avaliador automático de código que é acionado nas questões de programação dos questionários. A nota final de um questionário é uma média aritmética das maiores notas entre as tentativas de cada questão-problema. Esse parâmetro pode ser modificado pelo professor, por meio do ambiente de configuração do professor, assim como acontece nas avaliações da plataforma Moodle.

3.1. Experiências com o avaliador automático de código

Após a implementação e integração do avaliador de código do JavaTool na plataforma Moodle, dois importantes aspectos da avaliação foram considerados: usabilidade e eficácia educacional. A avaliação da usabilidade teve como referência os estudos de Kulyk et al. (2007), o qual afirma que está deve ser parte integrante do processo de um projeto e implementação sendo realizada em fases. Ao longo da evolução do projeto, a usabilidade foi avaliada das seguintes formas: i) avaliações informais, em que a opinião dos alunos foi capturada após o uso do sistema (Mota et al. 2009); ii) avaliações heurísticas, realizadas pelo professor especialista, sobre o uso dos recursos interativos e as características específicas para sistemas de visualização e simulação (Mota et al. 2008) (Moreira et al. 2009) (Mota et al. 2009); iii) estudos observacionais, em que os projetistas e avaliadores atentam para o uso do sistema e registraram aspectos importantes sobre este, verificando pontos a melhorar ou a especificação de novos requisitos (Mota et al. 2009); e iv) questionários, por meio dos quais os alunos foram consultados sobre diferentes aspectos do sistema.

Quanto à eficácia educativa, na tentativa de comprovar as hipóteses de Hundhausen et al. (2002) de que o esforço dedicado pelos alunos às tarefas relacionadas com a visualização é mais importante que o conteúdo das visualizações do JavaTool, foram realizados estudos observacionais e aplicação de questionários em duas turmas (29 e 47 alunos respectivamente) da disciplina de algoritmos e programação do curso de Ciência da Computação. Os questionários foram aplicados no começo do curso, com a finalidade de capturar os conhecimentos iniciais dos estudantes quanto ao desenvolvimento da lógica e capacidade de organização para problemas estruturados, e ao final do curso, com o intuito de avaliar a motivação quanto ao uso do ambiente de aprendizagem e a interface das tecnologias utilizadas.

De fato, por meio da interface do professor, especificamente desenvolvida para acompanhar as tentativas dos estudantes em cada questão (Figura 1), é possível observar que, mesmo nas situações em que a solução resolve o problema (notas acima de sete), pelo menos 70% das soluções passam pelo processo de "refinamento". O uso do avaliador incentiva o estudante a melhorar suas soluções e a refletir sobre o que considera um "programa correto". Além disso, o *feedback* automático reduz o tempo de espera do estudante pela avaliação de sua tarefa, ao mesmo tempo em que minimiza a sobrecarga do professor com as atividades de mediação.

O modelo de avaliação automática utilizado foi proposto por Moreira e Favero (2009) e combina o uso de uma avaliação da complexidade do código por meio da

técnica estatística de Regressão Linear Múltipla com indicadores de complexidade, aliada a um testador de código por entrada/saída (Mota et al 2009). Se não houver erro de compilação, o sistema avalia por complexidade e obtém um valor para a nota do estudante, que pode ser modificada dependendo do resultado da avaliação por testes de entrada/saída. O estudante pode consultar o histórico de tentativas que realizou em cada questão, em interface similar à do professor (Figura 1).

Pergunta	Texto da pergunta	Nome completo	Histórico das respostas	Melhor nota
1	Criar um programa que leia 2 números do tipo double. Escreva no console o menor valor entre esses números, use print(menorvalor). Se os valores foram iguais, use print("iguais").			
		Aluno 1	#1(9.7), #2(0.0), #3(0.0), #4(0.0), #5(0.0), #6(0.0), #7(0.0), #8(0.0), #9(0.0), #10(0.0), #11(0.0), #12(0.0), #13(0.0), #14(9.7), #15(9.7), #16(9.7), #17(0.0), #18(9.5), #19(0.0), #20(0.0), #21(0.0), #22(9.5), #23(9.4), #24(9.4), #25(9.4), #26(9.7), #27(0.0),	9.7
		Aluno 2	#1(4.8), #2(4.6), #3(0.0), #4(0.0), #5(0.0), #6(0.0), #7(3.1), #8(3.1), #9(9.4),	9.4
		Aluno 3	#1(6.4), #2(9.5),	9.5
		Aluno 4	#1(0.0), #2(6.3), #3(9.4),	9.4

Figura 1. Interface do professor (avaliação por questão) para acompanhamento dos questionários de programação

Os resultados dos estudos observacionais revelaram que pelo menos 40% dos estudantes utilizaram o recurso de "acelerar" as visualizações em uma tentativa de "passar mais rápido pela visualização" para receber resposta (*feedback)* do avaliador automático. Em resposta aos questionários, 95% dos estudantes declararam que o avaliador de código é útil para fornecer um *feedback* mais rápido para soluções incompletas ou não adequadas ao problema, sendo que 49% dos estudantes alegaram que o visualizador se apresentou útil para "rastrear" a execução nessas situações. Esses resultados impulsionaram a pesquisa (Brito et al., 2011) em direção ao do potencial do *feedback* e das possibilidades de ampliação desse potencial por meio do *feedback* colaborativo, no qual o estudante não apenas recebe o feedback, mas também o fornece.

3.2 O subsistema de *feedback* colaborativo: foco no processo de geração do feedback.

Na proposta de Mota et al. (2009), o *feedback* podia ser gerado pelo professor ou por monitores, por meio de comentários na solução do estudante. Com o avaliador automático de código, o próprio ambiente fornece um tipo de *feedback.* Os experimentos realizados comprovaram aumento no interesse dos estudantes no processo de refinamento da solução.

A partir desses experimentos, os resultados da pesquisa promoveram novas mudanças na arquitetura original de Mota et al. (2009) como uma estratégia de explorar os benefícios do *feedback* colaborativo e colocar o estudante ativo no processo de avaliação das soluções. Na nova concepção, considera-se o estudante no centro do

processo de avaliação, ou seja, o estudante não apenas recebe o *feedback*, mas o produz ele mesmo, como um exercício em que a finalidade principal é estimular a autoavaliação.

Assim, em uma sessão de exercícios, os estudantes recebem o *feedback* automático do avaliador, e após a conclusão do questionário de programação, são convidados a avaliar as soluções cadastradas por outros estudantes. A partir da especificação do componente de autoavaliação, a arquitetura foi adaptada para contemplar novas funcionalidades do subsistema de *feedback* colaborativo (Figura 2).

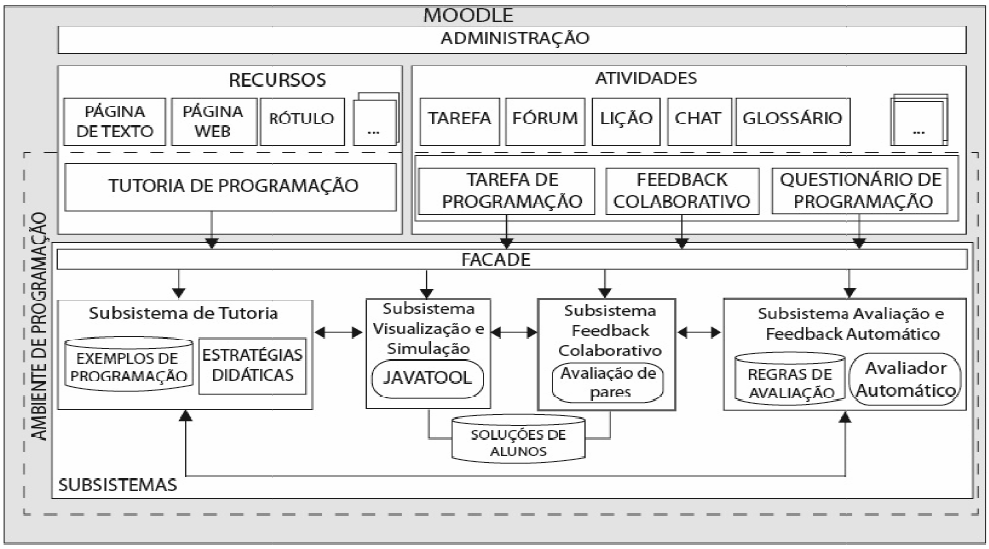

Figura 2. Arquitetura da integração do ambiente de programação ao Moodle, em destaque o subsistema (adaptada a partir de Mota et al., 2009)

Posteriormente, as mensagens de *feedback* podem ser socializadas pela turma, embora isso ainda não seja possível antes de uma rigorosa avaliação sobre os efeitos do *feedback* compartilhado nas crenças e autoconceito (Bandura 1986) do estudante. Para o estudante que recebe o *feedback*, a mensagem pode reforçar, concorrer ou conflitar com a própria interpretação do estudante sobre os resultados alcançados.

No projeto consideramos a interação entre estudantes como uma forma de oferecer ao estudante os mecanismos necessários para compreender claramente (Sadler 1998): (1) quais são os objetivos da disciplina, ou seja, o que é um considerado um bom desempenho, internalizando uma meta de aprendizagem a ser alcançada; (2) como o desempenho atual se relaciona com o desempenho esperado (para isso, os estudantes devem ser capazes de comparar o desempenho alcançado e o esperado); (3) como agir para reduzir ou eliminar a lacuna entre o desempenho atual e o esperado.

Para que os estudantes possam reduzir a lacuna entre os objetivos educacionais esperados e os objetivos alcançados, Sadler (1998) sugere que os estudantes devem possuir "algumas das competências de avaliação do professor". Para alguns autores, como Yorke (2003) e Boud (2000), esta observação levou à conclusão de que, além de melhorar a qualidade das mensagens de *feedback*, é preciso investir esforços na construção de competências de autoavaliação. Os argumentos de Sadler (1998) explicam o fato de que estudantes em melhores condições de autoavaliar suas produções alcançam significativos progressos, mesmo quando o *feedback* fornecido é insuficiente.

Com base nessa concepção, propomos incorporar ao ambiente de ensino em programação uma nova funcionalidade, na qual o estudante é convidado a fornecer um *feedback* para soluções produzidas por outros estudantes.

Esta avaliação é realizada selecionando-se aleatoriamente um problema dentro do mesmo nível de dificuldade explorado pelo estudante. Por meio desse subsistema, o estudante pode avaliar e fornecer um *feedback* baseado no seu julgamento da qualidade da solução apresentada. O estudante recebe imediatamente uma informação sobre sua avaliação, se foi correta ou não. As avaliações são fornecidas para os autores das soluções apresentadas, juntamente com o *feedback* colaborativo, conforme a figura 3.

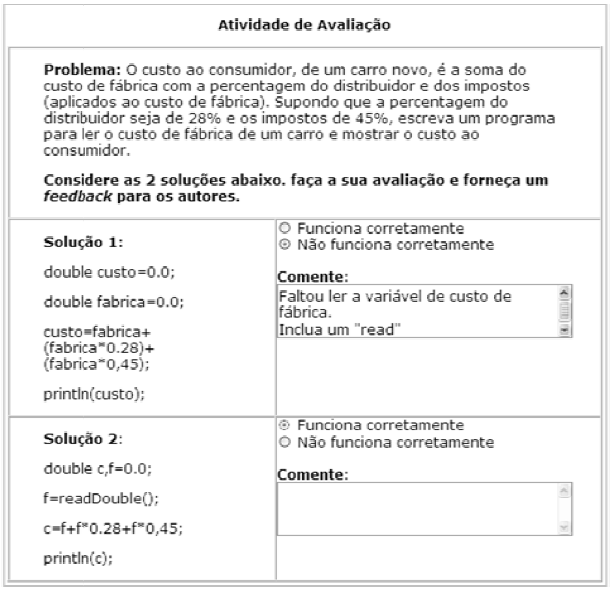

Fig. 3. Atividade de *feedback* colaborativo (avaliação de pares)

4. Discussões

Segundo Naps et al. (2003), quanto mais alto o nível de envolvimento do estudante, maiores são os benefícios educacionais. Utilizando a taxonomia de Naps et al. (2003), que especifica níveis para as diferentes formas de interagir com as tecnologias propostas, o ambiente proposto permite alcançar os seguintes níveis: a) visualização: o estudante pode visualizar uma animação passivamente, mas também exerce controle sobre a direção e o ritmo da animação, controles de velocidade e usando diferentes janelas, com uma apresentação visual ou acompanhando com explicações textuais ou fonéticas; b) questionamentos: o estudante é convidado a responder a questões quanto às visualizações apresentadas pelo sistema, podendo utilizar a visualização como recurso para responder a perguntas; c) modificação da visualização: permite que o aluno possa alterar a entrada do algoritmo a fim de explorar o seu comportamento em diferentes casos; e, d) construção: os estudantes constroem suas próprias visualizações dos algoritmos em estudo. Hundhausen et al. (2002) identificaram duas formas principais em que os alunos podem construir visualizações: geração direta, como no caso do JavaTool e construção manual.

O último nível da taxonomia trata da apresentação, ou seja, implica em apresentar a visualização para uma plateia, promovendo discussão e *feedback* entre pares. Esse nível pode ser alcançado a partir da adoção de uma metodologia em que o

estudante apresenta para a turma, registrando a sua solução, postando e comentando nos fóruns ou outros mecanismos de interação disponíveis. Assim, como forma de promover as competências de autoavaliação, os esforços da pesquisa concentraram-se na proposta do subsistema de *feedback* colaborativo, evitar formas de comparação social e favorecer os mecanismos que contribuam para a percepção do estudante quanto ao seu próprio desempenho.

Com o avanço da pesquisa e incorporação de novos componentes, é necessário ampliar as estratégias de avaliação do ambiente de aprendizagem, considerando tanto a usabilidade quanto a eficácia educacional porque nos estudos observacionais, assim como nos questionários, o ambiente e as tarefas executadas pelos alunos são parcialmente controlados pelos avaliadores.

5. Considerações finais e Trabalhos Futuros

Com uma proposta inicial de avaliar as soluções dos estudantes e reduzir a sobrecarga de trabalho do professor no acompanhamento das aulas de laboratório de algoritmos e programação, os esforços de pesquisa e desenvolvimento do JavaTool estavam centrados na produção de um *feedback* rápido e de qualidade que incentive os estudantes a refinar suas soluções. Além disso, a concepção de "transmissão de *feedback"* concentra no professor o esforço para produzir esse *feedback* o que gera uma sobrecarga de trabalho crescente, conforme o número de estudantes e turmas, o que nem sempre garante a eficácia de seu apoio no processo de ensino-aprendizagem (Brito et al., 2011).

A pesquisa apresentou um avanço significativo quando incorporou o avaliador automático ao ambiente que já apresentava as funcionalidades de simulação e visualização de código. Com a os experimentos observacionais, foi possível constatar um significativo interesse dos estudantes nos instrumentos de avaliação e no *feedback* do avaliador automático. O uso da visualização e simulação do código apresentou maior utilidade nas situações em que o estudante não alcançava o resultado esperado na avaliação. É possível considerar que sem o avaliador de código, o visualizador estava, de certo modo, sendo executado como uma ferramenta de auxílio ao processo de autoavaliação. Portanto, os componentes da arquitetura se complementam nas funções de construir, executar, demonstrar e avaliar as soluções do estudante.

Embora a pesquisa ainda demande novos testes e experimentos com o subsistema (de avaliação colaborativa), este artigo traz uma contribuição importante à medida que discute a avaliação automática não apenas como um instrumento de avaliação, com a finalidade de reduzir a sobrecarga do professor, mas como um componente que apóia os processos de autorregulação da aprendizagem, auxiliando também com a identificação de requisitos para novas funcionalidades.

Incorporar o subsistema em outras atividades da plataforma Moodle, além do ambiente de programação que envolve o JavaTool e o avaliador Automático. Finalmente, como requisito não funcional, a meta é manter o ambiente portável para que novos módulos possam ser adicionados sem que haja problemas com as versões da plataforma Moodle.

Referências

Bandura, A. (1986). "Social Foundantions of Thought & Action – A Social Cognitive Theory". *Englewood Cliffs*: Prentice Hall, 1986.

Barr, R. B. e Tagg, J. (1995). "A New Paradigm for Undergraduate Education", Change, 27(6), 13-25.

Boud, D. (2000). "Sustainable assessment: rethinking assessment for the learning society". In: *Studies in Continuing Education*, 22(2), 151-167.

Brito, S. R., Silva, A. S., Tavares, O.L., Favero, E.L., Francês, C. R. L. "Computer Supported Collaborative Learning for helping novice students acquire self-regulated problem-solving skills in computer programming". In: *The 2011 International Conference on Frontiers in Education: Computer Science and Computer Engineering* (FECS'11), 2011, Las Vegas. Worldcomp, 2011. v.7.

Brown, M. H. (1988) "Algorithm Animation". MIT Press.

Butler, D.L. Winne, P.H. (1995). " Feedback and self-regulated learning: a theoretical synthesis", *Review of Educational Research*, 65(3),245-281

Hundhausen, C. D., Douglas, S. A., Stasko, J. T. (2002) "*A Meta-Study of Algorithm Visualization Effectiveness*". Journal of Visual Languages & Computing.

Hundhausen, C. D., Douglas, S. A., Stasko, J. T. (2002) "A Meta-Study of Algorithm Visualization Effectiveness". *Journal of Visual Languages & Computing*.

Jeliot 3", Proceedings of the Advanced Visual Interfaces.

Kulyk, O., Kosara, R., Urquiza-Fuentes, J., Wassnick, I. (2007). "Human-centered visualization environments". *Lecture Notes in Computer Science*, vol. 4417, Chapter Human-Centered Aspects. Springer-Verlag, 13–75.

Lea, S.J., Stephenson, D. & Troy, J. (2003). "Higher education students' attitudes to student-centred learning: beyond 'educational bulimia'", In: *Studies in Higher Education*, 28(3), 321-334.

Menezes, C. S., Tavares, O.L., Nevado, R.A., Cury, D. (2008) "Computer Supported Co-operative Systems to support the problem solving - a case study of learning computer programming". In: *The 38th Annual Frontiers in Education (FIE) Conference*, 2008, New York. v. 1.

Moodle (2010). *Course Management System*. Disponível: <http://moodle.org/>. Acesso em: Maio. 2010.

Moreira M. P., Favero, E. L. (2009). "Um Ambiente Para Ensino de Programação com Feedback Automático de Exercícios". In: *Workshop Sobre Educação em Computação - Anais do XVIII Congresso da Sociedade Brasileira de Computação*. Belém-Pará: SBC, 2008.

Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M. (2004) "*Visualizing Programs with*

Mota, M. P., Brito, S. R., Moreira, M. P., Favero, E. L. (2009*). Ambiente Integrado à Plataforma Moodle para Apoio ao Desenvolvimento das Habilidades Iniciais de Programação"*. In: XX Simpósio Brasileiro de Informática na Educação (SBIE 2009). Florianópolis.

Mota, M. P., Pereira, L. W. K., Favero, E. L. (2008). "Javatool: Uma Ferramenta Para Ensino de Programação". In: *Workshop Sobre Educação em Computação, Belém-Pará. Procedings of XVIII Congresso da Sociedade Brasileira de Computação*. Porto Alegre-RS: Sociedade Brasileira de Computação.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Velázquez-Iturbide, J. Á. (2003). "Exploring the role of visualization and engagement in computer science education". *SIGCSE Bull*. 35, 2 (Jun. 2003), 131-152.

Perrenoud, P. (1999). "Avaliação. Da excelência à regulação das aprendizagens: entre duas lógicas". Porto Alegre: Artmed.

Polya, G., The Art of Solving Problems, Editora Interciência, 1st edition, 1978.

Proulx, V. K. (2000). "Programming Patterns and Design Patterns in the Introductory Computer Science Course". In: *Proceedings of the 31st SIGCSE*, pp. 7-12, Auxtin, TX, USA, march, 2000.

Rust, C., Price, M. and O'Donovan, B. (2003) "Improving students' learning by developing their understanding of assessment criteria and processes", In: *Assessment and Evaluation in Higher Education*, 28(2), 147-164.

Sadler, D.R. (1998) Formative assessment: revisiting the territory. In: *Assessment in Education*, 5(1), 77-84.

Santos, L. (2002). Auto-avaliação regulada: porquê, o quê e como? In Paulo Abrantes e Filomena Araújo (Orgs.), "Avaliação das Aprendizagens. Das concepções às práticas" (pp. 75-84). Lisboa: Ministério da educação, Departamento do Ensino Básico. http://www.educ.fc.ul.pt/docentes/msantos/textos/DEBfinal.pdf

Schunk, D.H. (1989). "Self-efficacy and cognitive skill learning". In: AMES, Carol & AMES, Russell (eds.) Research on Motivation in Education. *Goals and Cognititons*. New York: Academic Press, Inc., v. 3, p. 13-44, 1989.

Shang, Y., Shi, H., Chen, S. (2001). "An intelligent distributed environment for active learning". In: *ACM Journal of Educational Resources in Computing (JERIC)*. Vol. 1, No. 2, Summer 2001, Article 4, 17 pages. ISSN:1531-4278. New York: ACM Press.

Yorke, M. (2003). "Formative assessment in higher education: Moves towards theory and the enhancement of pedagogic practice". In: *Higher Education*, 45(4), 477-501.