Ambiente Integrado à Plataforma Moodle para Apoio ao Desenvolvimento das Habilidades Iniciais de Programação

Marcelle Pereira Mota[1, 3], **Silvana R. de Brito**[2], **Mireille Pinheiro Moreira**[1], **Eloi Luiz Favero**[1,2]

[1] Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Laboratório de Ensino a Distância (LabEAD)
Universidade Federal do Pará (UFPA) - Belém - Pará – Brasil

[2] Programa de Pós-Graduação em Engenharia Elétrica (PPGEE)
Universidade Federal do Pará (UFPA) - Belém - Pará – Brasil

[3] Bolsista CNPQ-Brasil

{cellemota,srossy,mireille,favero}@ufpa.br

Abstract. *The development of skills in computer programming requires significant effort from students makes solutions for problems proposed. This paper presents an exercise, simulation and evaluation environment to support the teaching-learning process in programming and algorithm courses. The integration of the environment to the Moodle platform increases its using benefits. The main advantages of environment are resulting from program viewer using, which contributes to the understanding of programs abstractions and the fast feedback provided by automatic evaluator. Furthermore, there is a decrement of teacher's overloading which allows the practice of new teaching strategies.*

Resumo. *O desenvolvimento de habilidades em programação de computadores exige um esforço significativo dos estudantes para construir soluções aos problemas propostos. Esse artigo apresenta um ambiente de exercício, simulação e avaliação para apoiar o processo ensino-aprendizagem em cursos de algoritmos e programação. A integração do ambiente à plataforma Moodle potencializa os benefícios de sua utilização. As principais vantagens do ambiente são advindas do uso do visualizador de programas, que contribui para a compreensão das abstrações dos programas e o rápido feedback fornecido pelo avaliador automático. Além disso, há diminuição da sobrecarga do professor permitindo a prática de novas estratégias de ensino.*

1. Introdução

O ensino de programação é fundamental em cursos de computação. As disciplinas de linguagens de programação, estruturas de dados e análise de algoritmos são matérias que devem compor o currículo dos cursos de graduação em computação de acordo com o currículo de referência da Sociedade Brasileira de Computação (SBC, 2003).

De acordo com Gomes *et al.* (2008), o mínimo que se espera é que o estudante seja capaz de desenvolver programas para resolver problemas reais simples, mas apesar

disso, são altos os níveis de reprovação em disciplinas de programação. Alguns dos obstáculos durante o ensino de programação são (Sajaniemi e Kuittinen, 2003): a sintaxe da linguagem e a dificuldade no entendimento de conceitos abstratos (construções formais de *loops*, ponteiros, *arrays*, etc.), que são vistos pela primeira vez. Em grande parte dos casos, quando o estudante percebe que não está evoluindo na disciplina, ele sente-se desmotivado e seu desempenho torna-se cada vez menor (Almeida, 2002).

Neste projeto, um ambiente de aprendizagem de programação é integrado à plataforma Moodle (Moodle, 2009). Para isso, foi criado um ambiente de aprendizagem que utiliza um visualizador de programas, testado e denominado JavaTool (Mota *et al.*, 2008), em exemplos, tarefas e questionários de programação. Além disso, quando o estudante submete suas soluções ao ambiente, é realizada uma avaliação, que pode ser automática, reduzindo a sobrecarga do professor no acompanhamento das soluções dos estudantes e agilizando o *feedback* necessário para o estudante reavaliar a sua solução.

Este artigo está organizado em quatro seções: a seção 2 apresenta desafios do processo de ensino-aprendizagem de programação e alguns trabalhos correlatos; a seção 3 descreve o ambiente proposto e a integração realizada; e, finalmente, a última seção apresenta considerações sobre o trabalho desenvolvido e direções para pesquisas futuras.

2. Desafios do Ensino de Programação e Trabalhos Relacionados

A aprendizagem em programação é um tema que vem sendo continuamente explorado no ensino da Computação. Um ambiente de aprendizagem de programação deve proporcionar suporte às etapas que compõem o processo de aprendizagem. É necessário, portanto, estabelecer quais são as etapas do processo de aprendizagem e que atividades podem ser integradas em cada uma delas para se atingir os objetivos pretendidos. As preocupações no planejamento dos cursos de programação consideram, pelo menos, as quatro dimensões (Bloom *et al.*, 1974): os objetivos educacionais esperados; as atividades e problemas propostos para alcançar os objetivos; a organização das atividades de forma que o estudante possa integrar os conhecimentos adquiridos, ou seja, o processo do curso; e a avaliação por meio de testes e outros procedimentos.

É importante observar que essas dimensões são interdependentes, ou seja, não se pode observar apenas o aspecto da avaliação sem considerar os objetivos esperados, as atividades e o processo do curso. Assim, para melhorar a qualidade do processo de ensino-aprendizagem é necessário compreender como as experiências de aprendizagem para programação estão relacionadas aos conceitos estudados e como podem ser organizadas para alcançar os objetivos esperados. Segundo Booth (1992), a pergunta central para a construção de tecnologias de apoio ao aprendizado da programação é "o que significa aprender a programar e o que é necessário fazer para alcançar esse aprendizado?" Para a autora, é necessário destacar qualitativamente as diferentes estratégias utilizadas pelos estudantes para compreender determinados aspectos da programação em estudo.

Para estabelecer a relação entre objetivos e experiências de aprendizagem, Stamouli e Huggard (2006) examinaram as concepções de programação que os estudantes iniciantes possuem e suas percepções sobre a "correção de um programa". Embora o estudo esteja relacionado com o paradigma da orientação a objetos, é possível relacionar as categorias citadas com o estudo da programação de modo geral. Em

direção ao significado da “aprendizagem de programação”, os autores identificaram níveis crescentes das habilidades desenvolvidas pelo estudante: conhecimento da sintaxe, compreensão das construções de programação, aprendizado da escrita de programas, aprendizado do pensamento lógico da programação, resolução de problemas e aquisição de novas habilidades. As três primeiras categorias estão centradas na linguagem de programação e suas construções, enquanto que o pensamento lógico da programação e a capacidade de resolução de problemas estão relacionados à habilidade da lógica individual para projetar uma solução. A última categoria considera a aprendizagem de programação útil para a aquisição de novas habilidades que extrapolam os limites de um curso de programação para outras áreas de conhecimento.

2.1 Acompanhamento dos estudantes nas atividades de programação

O processo de ensino-aprendizagem de programação envolve inúmeras dificuldades, relacionadas aos diferentes níveis de conhecimento dos estudantes. Uma das principais dificuldades observadas é o tempo reduzido que é dedicado para as atividades práticas com o acompanhamento do professor/monitor. É comum que as atividades sejam realizadas, sem a necessária análise da solução pelo estudante. Como resultado, soluções que poderiam ser refinadas são abandonadas como um “problema resolvido”. Segundo Stamouli e Huggard (2006), na aprendizagem de programação, embora a compreensão da correção lógica do programa seja importante, a análise da solução pelo estudante também é fundamental porque afeta diretamente na abordagem adotada para construir sua solução.

Quadro 1. Categorias de descrição para o entendimento de “programa correto” (Stamouli e Huggard, 2006)

No.	Categorias	Descrição
1	Sintaxe correta	O programa é percebido como correto quando compila sem erros, ou seja, quando executa, independente da sua funcionalidade.
2	Funcionalidades implementadas	Além de estar correto sintaticamente, o programa atende os requisitos da especificação do problema.
3	Correção do projeto	Além dos aspectos das categorias anteriores, o programa deve estar estruturado corretamente. Nesta categoria, os estudantes buscam técnicas para tornar a solução reusável, manutenível e extensível.
4	Desempenho e validação de entradas e saídas	Adicionalmente, o programa deve tratar erros como entradas inválidas e deve ser otimizado em termos de tamanho de código e rapidez de execução.

No Quadro 1, as duas primeiras categorias estão focadas mais no problema e na correção do código para o problema, enquanto que as duas últimas são importantes para incorporar elementos não-funcionais relacionados à qualidade da solução. A análise da solução pelo estudante contribui para alcançar uma solução sintática e semanticamente correta, além de possuir outros requisitos de qualidade, como desempenho e manutenibilidade. Neste trabalho, as categorias 1 e 2 são contempladas pelo visualizador de programas e a categoria 3 e 4 são atendidas pelo sistema de avaliação. Os resultados obtidos com a categoria 4 ainda são insuficientes para inferir que o uso do ambiente facilite o aprendizado dessas habilidades avançadas de programação. Portanto, o escopo deste trabalho está centrado nas categorias 1, 2 e 3, direcionado para o aprendizado das habilidades iniciais de programação.

Durante as práticas de programação, é comum observar uma sobrecarga de trabalho do professor para as atividades de avaliação, esclarecimento de dúvidas, ou

mesmo para organização da turma para programação em pares, valorizando atividades colaborativas. O atraso no *feedback* do professor para uma solução do estudante pode contribuir para a desmotivação no aprendizado da programação. Para prover um *feedback* rápido, o professor pode ser auxiliado por monitores, mas é difícil dispor de monitores suficientes.Portanto, um sistema de apoio ao ensino-aprendizado de programação com gerador automático de *feedback* deve orientar 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. Nesse contexto, a avaliação de atividades de laboratório de programação apresenta desafios e oportunidades inovadoras. Como desafios, temos: Como avaliar rapidamente os programas dos estudantes e fornecer *feedback* imediato, para melhor aproveitar o tempo de atividades em laboratório? Como cobrir os diferentes tipos de cognição na taxonomia de Bloom *et al.* (1974)?

2.2 Trabalhos relacionados: ambientes, visualizadores e simuladores

A habilidade de programação, para iniciantes, não pode ser adquirida sem um significante esforço nas atividades práticas de laboratório. Entretanto, pesquisadores do ensino de programação (Venables e Haywood, 2003; Truong *et al*., 2004) expõem a dificuldade de fornecer um *feedback* em tempo oportuno e que permita ao estudante pensar sobre a qualidade do código sendo desenvolvido, forçando-o a examinar soluções alternativas em busca de um melhor código.

Nessa direção, surgiram os programas e sistemas de visualização de programas e algoritmos, cujos benefícios justificam a disseminação das pesquisas nessa área (Stasko *et al*. 1993). Por outro lado, em um dos mais importantes estudos sobre a efetividade desses sistemas (Hundhausen *et al*., 2002) concluiu que a forma com que os estudantes utilizam as visualizações é mais importante do que as animações e imagens em si. O estudo mais recente de avaliação desses sistemas, realizado por Urquiza-Fuentes e Velázquez-Iturbide (2009) utiliza a taxonomia de Price *et al*. (1998): Visualização de algoritmos (VA), para animar ou apresentar estaticamente o algoritmo; Visualização de Programas (VP), para animar ou apresentar estaticamente o código ou as estruturas de dados do programa; Sistemas baseados em Script (SBS), onde o usuário seleciona os trechos de código que deseja visualizar; Sistemas de Interface (SI), que não geram visualização, apenas interagem com o usuário (em alguns casos invocando os SBS), e; os Sistemas Baseados em Compiladores, que inserem as ações de visualização automaticamente, simplificando o seu uso, mas limitando as ações do usuário.

Na categoria dos VA, Almeida *et al.*, (2002) descrevem o ambiente AMBAP e Silva e Favero (2005) descrevem o ambiente VisualPseudo. A utilização de pseudocódigo apresenta a desvantagem da falta de padronização, onde um programador pode não entender a lógica de um programa escrito por outra pessoa (Halstead, 2007). Noutra linha, Gomes e Mendes (2000) descrevem o ambiente SICAS, centrado em fluxogramas com símbolos diagramáticos, focando mais a estrutura dos algoritmos. Os VP vêm sendo desenvolvidos e avaliados há algum tempo. A visualização de programas foca na representação gráfica da execução de um programa e seus dados. A representação visual de dados e da seqüência de execução de um programa pode proporcionar aos usuários valiosa informação em muitos aspectos do desenvolvimento e

da execução de um programa, como depuração ou comparação entre diferentes algoritmos em termos de desempenho, por exemplo (Ellershaw e Oudshoorn, 1994).

Uma das abordagens da visualização de programas é a animação de algoritmos, que consiste em produzir visualizações animadas de algoritmos, geralmente durante a execução de um programa. Um dos sistemas pioneiros para visualização de programas é o BALSA (Brown, 1988). Nesta linha, o JEliot (Moreno *et al*., 2004) permite a visualização e animação da execução de um código em Java, inclusive mostrando as estruturas de dados. O JavaTool (Mota *et al.*, 2008) é semelhante ao JEliot, pois ambos animam código Java, porém possui representação gráfica diferente, animação de código com sintaxe simplificada e é mais indicado para o ensino de programadores iniciantes, pois não inclui conceitos de orientação a objetos.

Portanto, a motivação para o desenvolvimento deste trabalho foi criar um ambiente para ensino de conceitos introdutórios e desenvolvimento de habilidades iniciais de programação. Utilizando uma sintaxe simplificada da linguagem Java, o estudante pode editar código Java, compilar, depurar, visualizar dinamicamente a execução da sua solução e obter um rápido *feedback* por meio do avaliador automático.

3. Ambiente de aprendizagem de programação no contexto do Moodle

O Moodle (2009) é uma plataforma para gerenciamento de cursos (SGC), embasado na pedagogia sócio-construcionista (Papert, 1986). Como um *software Open Source*, multiplataforma, distribuído sob a licença *General Public Licence* (GPL), a plataforma Moodle facilita a integração de novas tecnologias educacionais, o que simplifica o uso para estudantes e professores porque permite reunir na estrutura de um curso, diferentes recursos educacionais, mantendo um registro das participações.

Nesta fase, o foco do projeto está na aprendizagem das habilidades iniciais de programação. Embora a linguagem utilizada pelos estudantes seja Java, os conceitos de orientação a objetos não estão incluídos nesse primeiro momento de aprendizado. O ambiente é formado pelos sistemas de Tutoria, de Simulação e de Avaliação, que fazem uso do simulador de animação e do avaliador automático de programas.

3.1. Integração do Ambiente à plataforma Moodle

A integração permite que o ambiente de programação seja acessado e utilizado por meio da plataforma Moodle. O Moodle possibilita a inserção de recursos e atividades, onde os recursos disponibilizam conteúdo e atividades promovem interação com o usuário. O sistema de Tutoria é utilizado como um tipo de recurso e os sistemas de Simulação e Avaliação de Programas são tipos de atividades do Moodle (Figura 1).

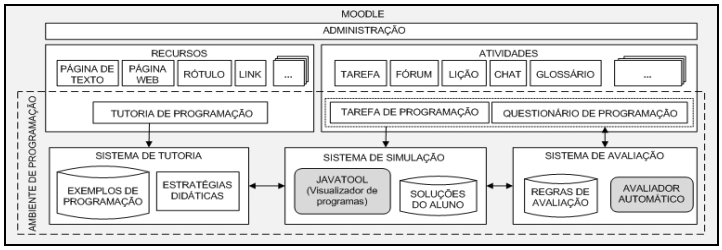

Figura 1. Arquitetura da integração do ambiente de programação ao Moodle

Foi desenvolvido um novo tipo de recurso denominado tutoria de programação e dois novos tipos de atividades (tarefa e questionário de programação). Nas tutorias de programação, o professor armazena o código exemplo e o estudante visualiza a animação do código exemplo. A tarefa de programação é incluída pelo professor como um problema que deve ser resolvido pelo estudante. O questionário de programação consiste de um conjunto de questões-problemas de programação, selecionadas pelo professor. O estudante pode consultar as questões e o histórico de suas tentativas com as respectivas notas (Figura 2). A nota final é uma média aritmética das maiores notas entre as tentativas de cada questão-problema. Esse parâmetro pode ser modificado pelo professor, através do ambiente de configuração do professor, assim como acontece nas avaliações da plataforma Moodle. O sistema de Avaliação permite que o professor avalie as respostas dos estudantes manualmente ou possibilita um *feedback* imediato por meio do avaliador automático.

Pergunta	Texto da pergunta	Histórico das respostas	Melhor nota	
1	Descubra o maior de três números.	#1(8), #2(7),	8	Responder
2	faça o fatorial de n.	#1(4),	4	Responder
3	Imprima um numero aleatorio entre 1 e 10.	#1(3), #2(5), #3(9),	9	Responder

Figura 2. Visualização inicial no questionário de programação

3.1.1 Sistema de Tutoria

No decorrer de um curso de programação, é importante que o estudante tenha acesso a materiais didáticos básicos como conteúdos e exemplos. O sistema de Tutoria gerencia os exemplos com códigos-fonte de programas armazenados na base de exemplos, que são exibidos como recursos dentro da plataforma. Os exemplos são classificados de acordo com estratégias didáticas pré-estabelecidas formando uma base de exemplos de programação. Esse sistema tem função semelhante ao módulo tutor apresentado por Branco Neto e Schuvartz (2007), onde o estudante participa do ambiente visualizando exemplos que estão disponíveis de acordo com o seu grau de conhecimento e habilidade em programação.

O professor cadastra os exemplos ou pode salvar uma solução desenvolvida pelo estudante na base de exemplos. Os exemplos estão disponíveis como conteúdos e podem ser utilizados de acordo com os critérios do professor ou por livre exploração do estudante. As questões de programação podem ser relacionadas aos exemplos, conduzindo o estudante a correlacionar os problemas e suas soluções. Assim, se o estudante percebe, na simulação, um erro de sintaxe na declaração de uma estrutura de repetição *do-while*, pode buscar um exemplo relacionado àquela estrutura de repetição na base de exemplos. Outras hipóteses podem ser viabilizadas, pois o módulo é flexível e adaptável, já que segue as orientações de desenvolvimento para o Moodle (2009).

3.1.2 Sistema de Simulação

Um estudo baseado no funcionamento dos hemisférios cerebrais confirma a hipótese de que a visualização da animação de algoritmos pode contribuir para facilitar o entendimento (Springer e Deutsch, 1985). De acordo os autores, o hemisfério esquerdo processa informações orais e lógicas enquanto o hemisfério direito processa as informações visuais e espaciais. Por isso, ao ler determinado código fonte e visualizar uma representação gráfica desse código são estimulados os dois hemisférios cerebrais,

aumentando a chance de uma melhor compreensão acerca do código.

A simulação ocorre por meio da animação do código-fonte. Pode ser a partir de um exemplo ou de uma solução do estudante para uma tarefa ou um questionário de programação. Na Figura 3, pode-se observar uma tela do funcionamento do simulador no Moodle, após a resolução de uma questão-problema em um questionário de programação.

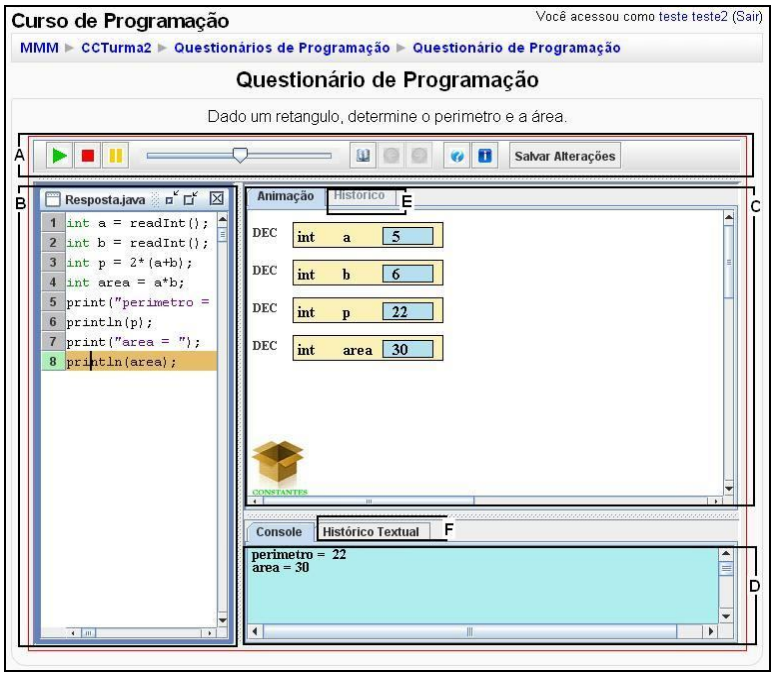

Figura 3. Visualização do JavaTool na plataforma Moodle

A área ‘A’, demarcada na Figura 3, representa a barra de ferramentas. Nesta, estão localizados o menu de animação e histórico, além dos botões de ajuda, informações e armazenamento do código escrito pelo usuário. O editor é representado pela área ‘B’, onde se observa o uso do recurso de cor nas palavras reservadas da linguagem. A área ‘C’ corresponde ao painel de animação, onde as estruturas são criadas e podem ser visualizadas. A área ‘D’ apresenta o resultado da saída do algoritmo executado. As áreas ‘E’ e ‘F’ (desabilitadas) são responsáveis pela representação gráfica e textual das operações realizadas durante a simulação.

O simulador possibilita a animação textual e gráfica do código mostrando o seu comportamento na execução, para que o estudante possa compreender corretamente a lógica do programa. Dentre os recursos da linguagem Java disponíveis estão: tipos primitivos, *arrays* unidimensionais, estruturas de seleção e repetição, alguns métodos da classe *Math* e a criação e chamada de métodos simplificados.

3.1.3 Sistema de Avaliação

O sistema de avaliação permite a avaliação automática ou manual, com lançamento de notas e comentários, conforme os recursos do Moodle. O modelo de avaliação automática implementado no ambiente de programação foi proposto por Moreira e Favero (2009). Esse modelo combina o uso de uma avaliação da complexidade do

código através da técnica estatística de Regressão Linear Múltipla com indicadores de complexidade, aliada a um testador de código por entrada/saída.

Os indicadores utilizados pelo avaliador por complexidade de código são métricas da Engenharia de Software. A técnica consiste em extrair da solução do estudante o valor obtido com a aplicação de cada uma das métricas descritas por Moreira e Favero (2009). Esses valores são submetidos ao modelo de Regressão Linear Múltipla, que consiste em uma equação linear obtida previamente através de treinamento em uma base de testes. Neste caso, as métricas são as entradas da fórmula que calcula diretamente a nota do estudante. Em seguida, é verificado se a solução retorna o resultado esperado para as entradas previamente informadas.

Assim, para o correto funcionamento do avaliador, o professor deve cadastrar uma solução considerada "resposta-modelo" para o problema. Assim, 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. A nota final é então exibida na tela (Figura 4).

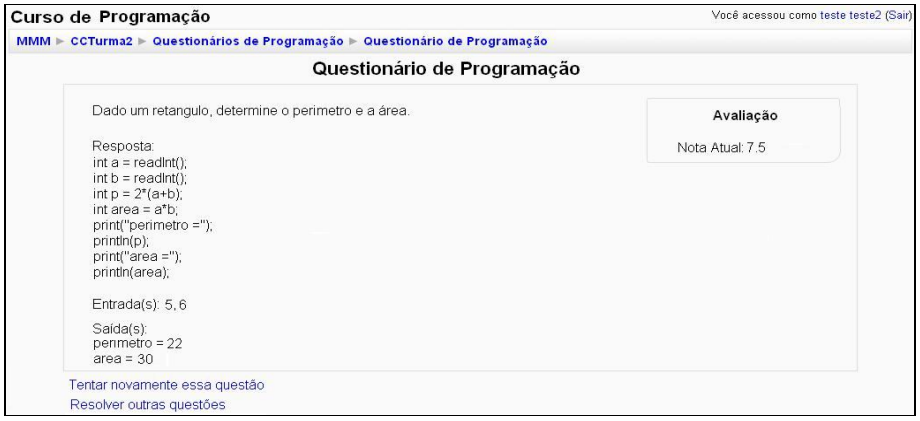

Figura 4. Visualização da avaliação automática da resposta do estudante

Ao utilizar a avaliação automática, o professor visualiza os resultados, verificando o sucesso dos estudantes ou a necessidade de uma intervenção (Figura 5). A agilidade do *feedback* permite que o professor planeje melhor as atividades colaborativas ou aproveite melhor a atuação dos monitores, de acordo com o mapa de evolução dos estudantes durante as práticas de laboratório.

Figura 5. Visualização das respostas dos estudantes por questão

4. Considerações Finais e Trabalhos Futuros

Conceitos de Programação são fundamentais para o estudo da Ciência da Computação e para o Desenvolvimento de Sistemas. Esses aspectos são tão críticos na formação dos

estudantes que os desafios da programação são apresentados no início da vida acadêmica dos cursos de graduação, com a expectativa de que os estudantes adquiram essas habilidades o mais rápido possível. Entretanto, alguns estudantes encontram muitas dificuldades e isso tem um impacto negativo no seu desempenho do estudante durante as disciplinas mais avançadas do curso.

Na expectativa de contribuir com o índice de aproveitamento das disciplinas de algoritmos e programação, este projeto conjuga os resultados de uma ferramenta de visualização de programas e de um avaliador automático para fornecer um rápido *feedback* com uma plataforma que permite construir um ambiente interativo de aprendizagem. Embora as tecnologias utilizadas neste projeto tenham sido testadas isoladamente por Moreira e Favero (2009) e por Mota *et al.* (2008), a integração proposta potencializa o ambiente de acompanhamento das soluções para professores e alunos, ao mesmo tempo em que simplifica o seu uso por adotar o padrão da plataforma Moodle (2009), que é largamente utilizada em cursos de graduação.

Para o estudante, o novo ambiente deve contribuir para a compreensão das abstrações dos códigos elaborados ao mesmo tempo em que reduz o tempo do *feedback*. Para o professor, oportuniza a adoção de novas abordagens, principalmente aquelas colaborativas, porque fornece, mais cedo, um mapa com resultados ainda que parciais do desempenho dos estudantes. A prática com atividades colaborativas deve contribuir para a identificação de requisitos para novas funcionalidades, com foco nas demandas de atividades colaborativas e cooperativas, como a programação em pares.

Tratar níveis mais avançados do aprendizado de programação e incorporar o uso de outras linguagens, além de Java, também é meta do projeto. Para isso, outros métodos de avaliação automática de programas podem ser testados. 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

Almeida, E. S., Costa, E. B., Braga, J. D. H., Silva, K. S., Paes, R. B. E Almeida, A. A. M. (2002) "Ambap: Um Ambiente de Apoio ao Aprendizado de Programação". Workshop Sobre Educação Em Computação, Florianópolis.

Bloom, B. S., Engelhart, M. D., Furst, E. J., W. H. Hill, D. R. Krathwohl. (1974) "Taxonomia dos Objetivos Educacionais". Ed. Globo, Porto Alegre.

Booth, S., (1992) "*Learning To Program: A Phenomenographic Perspective*". Göteborg: Acta Univertsitatis Gothoburgensis. Disponível em: <http://www.ped. gu.se/biorn/phgraph/civil/graphica/diss.ab/booth.html>. Acesso: Ago. 2009.

Branco Neto, W. C., Schuvartz, A. A. (2007) "Ferramenta Computacional de Apoio ao Processo de Ensino-Aprendizagem dos Fundamentos de Programação de Computadores". Simpósio Brasileiro de Informática na Educação. São Paulo.

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

Ellershaw, S., Oudshoorn, M. (1994) "*Program Visualization*" - The State of the Art. Department of Computer Science, University of Adelaide. Disponível em: <http://citeseer.ist.psu.edu/ellershaw94program.html>. Acesso: Ago. 2009.

Gomes, A., Mendes, A., (2000) “Suporte à aprendizagem da programação com o ambiente SICAS”, V Congresso Ibero-Americano de Informática Educativa .

Gomes, A., Henriques, J., Mendes, A. J. (2008) “Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores”. In Educação, Formação e Tecnologias; vol.1 (a), p. 93-103.

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

Moodle (2009). Disponível Em: <http://moodle.org/>. Acesso Em: Ago. 2009.

Moreira M. P., Favero, E. L. (2009) “Um Ambiente Para Ensino De Programação Com Feedback Automático De Exercícios”. Workshop Sobre Educação em Computação.

Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M. (2004) “*Visualizing Programs with Jeliot 3*”, Proceedings of the Advanced Visual Interfaces.

Mota, M. P., Pereira, L. W. K., Favero, E. L. (2008) “Javatool: Uma Ferramenta Para Ensino de Programação”. Workshop Sobre Educação em Computação.

Papert, S. (1980) “*Mindstorms, children, computers and powerful ideas***”.** Basic Books.

Price, B., Baecker, R., And Small, I. (1998) “*An Introduction To Software Visualization*”. In Software Visualization, J. Stasko, J. Domingue, M. Brown, And B. Price Eds. Mit Press, Cambridge, Ma, 3–27.

Sajaniemi, J., Kuittinen, M. (2003) “*Program Animation Based On The Roles Of Variables*”. Proceedings of The 2003 ACM Symposium on Software Visualization.

SBC. (2003) “Currículo de Referência da Sociedade Brasileira de Computação para Cursos de Graduação em Computação e Informática”. Disponível Em: <http://www.sbc.org.br/index>. Acesso: Ago. 2009.

Silva, M. A. B., Favero, E. L. (2005) “Compiladores e Interpretadores Uma Abordagem Prática”. Semana Paraense de Informática.

Springer, S. P., Deutsch, G. (1985) “*Left Brain, Right Brain*”. W. H. Freeman And Company, New York.

Stamouli, I., Huggard, M. (2006) “*Object Oriented Programming and Program Correctness: The Students’ Perspective*”. Icer’06, Canterbury, United Kingdom.

Stasko, J., Badre, A., Lewis, C. (1993) “*Do Algorithm Animations Assist Learning*” *An Empirical Study and Analysis*”. In Proceedings of The Sigchi Conference on Human Factors In Computing Systems (Chi’93). 61–66.

Truong, N., Roe, P., Bancroft, P. (2004) “*Static Analysis Of Students' Java Programs*”, V.30, P.317-325, New Zealand ACM.

Urquiza-Fuentes, J. And Velázquez-Iturbide, J. (2009) “*A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems*”. Trans. Comput. Educ. 9, 2 (Jun. 2009), 1-21.

Venables, A., Haywood L. (2003) “*Programming Students Need Instant Feedback*!”. In Proceedings Of The Fifth Australasian Conference On Computing Education. Conferences In Research And Practice In Information Technology Series.