Técnicas para aumentar o Envolvimento dos Alunos na Aprendizagem da Programação

Paula Correia Tavares, Elsa Ferreira Gomes
ISEP, Instituto Superior de Engenharia do Porto
Porto, Portugal
{pct, efg}@isep.ipp.pt

Pedro Rangel Henriques
Algoritmi/DI, Universidade do Minho
Braga, Portugal
prh@dei.uminho.pt

Maria João Varanda Pereira
IPB, Instituto Politécnico de Bragança
Bragança, Portugal
mjoao@ipb.pt

Resumo

Neste artigo analisam-se em particular duas técnicas concebidas para apoiar o ensino da programação: a Animação de Programas e a Avaliação Automática de Programas. Com base na combinação destas técnicas e respetivas ferramentas, atualmente disponíveis, iremos enunciar duas possíveis abordagens. Serão apresentadas as conclusões retiradas de uma primeira experiência conduzida em sala de aula. Por fim, esboçaremos uma ferramenta que através da Web, implementará uma das abordagens propostas.

Palavras-chave: Motivação, Animação de Programas, Avaliação Automática de Programas, Feedback imediato.

1 Introdução

De acordo com (Hundhausen & Douglas, 2000), (Proulx, 2000) e muitos outros autores, incluindo as sucessivas notas realçadas no Computer Science Curricula Guideline da ACM/IEEE de 2013 (ACM/IEEE, 2013), e confirmado pela nossa longa experiência profissional a ensinar disciplinas de Introdução à Programação de Computadores, é certo que aprender a programar é uma árdua e complexa tarefa que levanta muitos desafios a qualquer uma das partes envolvidas. Uma das principais razões para o insucesso escolar dos alunos dos cursos de programação está na sua falta de motivação (Santos & Costa, 2006)(Ramos, 2013). Tal tem impacto direto na sua

capacidade de aquisição de conhecimento e consequentemente afeta os resultados do processo de ensino/aprendizagem, traduzindo-se em uma forte frustração para alunos e professores. São várias as razões que levam os alunos a falhar na aprendizagem da programação (Proulx, 2000) mas o que é certo e todos sabemos é que face à menor dificuldade na compreensão do enunciado, no desenvolvimento de um algoritmo ou no uso de uma linguagem de programação, os aprendizes desmotivam-se e desistem.

O objetivo do projeto de investigação em que se insere o trabalho aqui relatado visa: compreender as razões das reais dificuldades que se levantam ao processo de ensino/aprendizagem de Programação de Computadores; estudar abordagens suportadas por computador já criadas para combater esse insucesso; e sugerir formas de as combinar para tirar um maior partido no aumento do envolvimento dos alunos de modo a ultrapassar tais dificuldades.

Neste artigo analisam-se em particular duas técnicas concebidas para apoiar o ensino da programação: uma, mais antiga, a Animação de Programas que pretende tirar partido da nossa acuidade visual e no efeito da simulação para facilitar a compreensão dos programas e algoritmos subjacentes; e outra, muito mais recente, que aposta no recurso a sistemas de Avaliação Automática de Programas para incentivar os alunos a continuar providenciando-lhes feedback imediato após a conclusão da escrita de um programa. Com base na combinação destas técnicas e respetivas ferramentas, apresentadas na secção 2, iremos enunciar duas possíveis abordagens (secção 3). Iremos também resumir as conclusões já retiradas de uma primeira experiência conduzida em sala de aula (secção 4) e esboçaremos uma possível ferramenta que sustente, via Web, a utilização de uma das abordagens propostas (secção 5). A secção 6 fecha o artigo com um resumo e trabalho futuro.

2 Técnicas de apoio ao ensino da programação

De entre muitas técnicas que têm sido estudadas para ajudar no ensino da programação serão referidas neste contexto duas que nos pareceram particularmente interessantes: Animação de Programas e Avaliação Automática de Programas. Por motivos de espaço, a revisão destas duas áreas em que se baseia o nosso trabalho não poderá ser apresentada em detalhe, recomendando-se a leitura de (Tavares, Henriques & Gomes, 2015).

2.1 Animação de Programas

Segundo Maria João Pereira (Pereira, 2002), a animação de um algoritmo é um tipo de visualização dinâmica das principais abstrações expressas pelo mesmo. A sua importância reside na habilidade de retratar a essência da lógica do algoritmo.

Muitas vezes o professor tem a necessidade de usar representações visuais para ajudar os alunos a entenderem os algoritmos, o que na prática se concretiza muitas vezes mostrando o comportamento de programas. Por isso a animação de algoritmos é importante. A animação pode ser composta por um conjunto de visualizações de forma mais ou menos interativa. Ari Korhonen (Korhonen, 2003) explica que a questão fundamental é, no entanto, como se devem aplicar estes artefactos com a finalidade de ajudar os alunos a lidar com conceitos complexos. Segundo este autor, do ponto de vista pedagógico, será mais interessante a lógica e comportamento de um algoritmo do que os seus detalhes de implementação. Deve ser garantido que ocorreu, pelo menos, um nível de progresso na aprendizagem. Esta metodologia requer um ambiente onde se possa dar e obter feedback sobre o desempenho do aluno (Korhonen, 2003). Esta afirmação inspirou parcialmente a nossa proposta a apresentar mais à frente.

Neste contexto, existem várias ferramentas com o intuito de auxiliar os alunos a aprender a programar, que tentam introduzir conceitos básicos de programação através de um ambiente familiar e agradável. A seguir listam-se alguns dos mais conhecidos: BALSA (Saraiya, 2002), TANGO (Hughes & Buckley, 2004), Jeliot (Silva, D'Emery, Neto & Bezerra, 2009), Alma (Pereira & Henriques, 1999), SICAS (Mendes, Gomes & Marcelino, 2004), OOP-Anim (Santos, Gomes & Mendes, 2010) (Esteves & Mendes, 2003), VILLE (Rajala, Jussi, Erkki & Salakoski, 2007), JIVE (Lessa, Czyz & Jayaraman, 2011). Todas estas ferramentas, na realidade permitem não diretamente a animação do algoritmo mas, por razões óbvias, a animação de programas em diferentes linguagens de programação (C, Java, etc.).

A animação torna-se um facilitador do processo de aprendizagem visto que a apresentação de conceitos abstratos é mais didática, melhorando a qualidade do material de apoio à aula. Se desta forma se conseguir que o aluno adquira uma boa base, então o rendimento e desempenho aumentam proporcionando melhores resultados, melhores currículos e melhores profissionais (Santos & Costa, 2006).

2.2 Avaliação Automática de Programas (AAP)

É muito importante dar aos alunos a oportunidade de praticar assim como poder resolver exercícios de programação por si mesmos. Feedback imediato é crucial para a aquisição de conhecimento, independentemente da estratégia de aprendizagem adotada pois motiva os alunos. Por outro lado, o Feedback individual pode consumir muito tempo do professor com o risco de que os alunos possam não beneficiar dele no devido tempo (Queirós & Leal, 2015). Para resolver este problema, existem alguns sistemas de submissão online que suportam a avaliação automática de problemas de programação (Queirós & Leal, 2012). Diferentes estudos mostram que estes sistemas permitem aos alunos desenvolver autonomamente habilidades de programação e melhorar significativamente o seu desempenho (Verdú, Regueras, Leal & Queirós, 2011).

Novas ferramentas surgem assim para satisfazer os pedidos do avaliador e além de facilitar e permitir a sua utilização em atividades letivas, permitem ao aluno incorporar testes nos seus trabalhos. Os professores devem poder então selecionar os problemas que pretendem apresentar aos estudantes de acordo com seu nível de dificuldade (Verdú, Regueras, Leal & Queirós, 2011). Diferentes professores podem adotar diferentes estratégias, dependendo dos seus objetivos específicos e objetivos do curso, e principalmente do seu próprio estilo e preferências (Joy, Griffiths & Boyat, 2005). Como exemplos de ferramentas deste tipo, podem citar-se o Boss (Heng, Joy & Griffiths, 2005), o Mooshak (Leal, Silva, 2008) e o EduJudge (Verdú, Regueras, Leal & Queirós, 2011), em que o principal objetivo, além de testar as respostas dos alunos face a um conjunto de dados de entrada e dar uma classificação, é permitir ao avaliador motivar os alunos através de um feedback preciso e rápido.

2.3 Propostas

Como foi dito na introdução, a motivação principal do projeto aqui relatado reside na dificuldade do processo ensino/aprendizagem da programação e seu insucesso. Assim, o trabalho consiste na investigação de um conjunto de estudos relativos ao insucesso da programação e consequentemente na elaboração de propostas de solução com o objetivo de aumentar a motivação e a autoconfiança dos alunos dos cursos de Introdução à Programação.

A proposta de trabalho que se faz neste artigo tem como objetivo principal diminuir as dificuldades sentidas pelos alunos que se encontram numa fase de aprendizagem de programação. Estas dificuldades incentivaram desde há alguns anos a elaboração de novas ferramentas que permitem facilitar todo o processo de ensino e aprendizagem dos elementos

básicos da programação. Para tal, estudos experimentais, no âmbito escolar, são essenciais para o desenvolvimento ou uso de plataformas de ajuda à aprendizagem. Estas ferramentas serão elementos de apoio ao ensino para o aumento da capacidade dos alunos de resolução do seu problema. É importante ajudar os alunos na transição dos conhecimentos básicos, para uma visão de uma solução algorítmica, daí a importância destas ferramentas para apoiar este processo. O objetivo é fazer com que os alunos aumentem a sua capacidade de praticar com regularidade a programação, desde o primeiro dia, pois acreditamos que desta forma o seu sucesso escolar vai aumentar.

Face aos recursos informáticos atualmente disponíveis para Animação e Avaliação Automática de Programas propomos duas abordagens alternativas, a seguir descritas, com vista a agilizar o processo de ensino aprendizagem da programação.

Abordagem 1: O aluno é exposto diretamente à resolução com recurso à avaliação automática e seu feedback e depois à análise da solução correta com recurso à animação.

Para concretizar esta abordagem, o professor prepara para cada tema a apresentar, um conjunto de problemas relativos a esse tema de dificuldade semelhante. Para cada problema do conjunto, pede que o aluno analise o enunciado, desenvolva o algoritmo e o codifique passando a testá-lo com o sistema de AAP. Ao fim de um determinado tempo, o professor fornece a sua solução e pede aos alunos que analisem cuidadosamente recorrendo à ferramenta de animação procurando que assimilem o conhecimento daí derivado.

Abordagem 2: Primeiro o aluno é exposto à análise do problema e sua resolução, com o apoio da Animação; Depois passa a uma fase de auto resolução com recurso à avaliação automática e respetivo feedback rápido.

Por cada tema a ensinar, o professor prepare dois conjuntos de exercícios análogos. Para os exercícios do primeiro conjunto o professor discute o enunciado, esquematiza a resolução (esboça um algoritmo) e apresenta o programa que o resolve de modo a que o aluno possa fazer a sua animação e assim analisar/compreender a solução. Para os exercícios do segundo conjunto, após discutir o enunciado, o professor pede que os alunos o resolvam e testem a solução criada através de um sistema de avaliação automática. Num terceiro momento, o professor discute com os alunos o feedback recebido do avaliador.

Para aferir estas duas alternativas, que não são exclusivas, desenvolvemos um teste em sala de aulas seguindo a primeira abordagem e estamos agora a desenvolver uma ferramenta que materialize a segunda abordagem, conforme descrito nas secções seguintes.

3 Experiencia Realizada

Nesta secção será apresentada uma descrição do primeiro experimento realizado para aferir a abordagem 1.

Os principais objectivos para este primeiro experimento foram:

- compreender o comportamento dos alunos que enfrentam uma situação nova e diferente;
- observar se os alunos estão envolvidos e motivados;
- entender quais são as principais dificuldades dos alunos do primeiro ano, quando enfrentam uma tarefa de programação: a interpretação do enunciado, ou o desenvolvimento de algoritmos ou a sua codificação numa determinada linguagem;
- Verificar a eficácia da abordagem proposta.

Com base na descrição sucinta da abordagem em causa (Secção 3), elaborou-se o seguinte esquema para condução da sessão experimental:

Passo 1: O professor prepara três exercícios semelhantes para o tópico a ensinar.

Passo 2: Para o primeiro exercício, o professor deve analisar com os alunos o enunciado do problema, e, em seguida, pedir-lhes para resolver o problema e testar a solução produzida usando o Sistema de Avaliação Automático, SAA, selecionado. O professor também pode discutir com cada aluno o feedback recebido do SAA.

Passo 3: O professor fornece uma proposta de solução do exercício e o aluno deve utilizar o sistema de animação selecionado, para animar a execução do programa a fim de analisar com cuidado e entender a solução correta e seu comportamento.

Passo 4: Repetir os passos 2 e 3 para as restantes exercícios.

Esta abordagem pressupõe que o professor selecione uma ferramenta de animação eficaz e fácil de usar, bem como escolha um avaliador automático que além de amigável retorne um feedback que forneça um diagnóstico e, se possível, comente a qualidade do código. As ferramentas

escolhidas para os experimentos foram o Jeliot e o Mooshak, apresentados nos capítulos anteriores.

Para este caso específico relativo ao ensino dos tópicos introdutórios da Programação "sequências numéricas e estruturas de decisão e repetição", foram desenvolvidos três enunciados, com o cuidado de serem bastante semelhantes:

a) Desenvolver um programa em Java que leia uma quantidade não determinada de números inteiros positivos. Calcule e imprima a quantidade de números pares e ímpares assim como a média (float) dos valores pares. O número que encerrará a leitura será o zero.
b) Desenvolva um programa em Java que dado um número M e um número N, inteiros positivos, leia N idades mostrando todas as idades maiores que M. No final deve imprimir a média (float) das idades.
c) Dadas as temperaturas (float) de 6 dias do mês de Janeiro (valores entre -50° e 50°), mostre a temperatura máxima e mínima. Classifique também o mês como "frio" ou "quente" conforme teve mais dias com temperaturas negativas ou com temperaturas positivas (zero incluído). Em caso de igualdade considere que o mês é "frio".

Depois de decididas as ferramentas concretas a usar, o tema da aula experimental, e os exercícios para resolver, foi necessário escrever um plano cuidado da aula, para que todos os alunos conseguissem entender para o que são convidados a fazer e como devem proceder.

Antes de iniciar a experiência, o Jelliot foi instalado em todos os computadores da sala de aula, e o Mooshak foi configurado e preparado num servidor fornecido pelo Departamento de Ciências de Computadores da FCUP, a fim de ser acedido pelos alunos via Internet (Mooshak é uma ferramenta acessível on-line).

O experimento foi feito com 28 alunos do 1º ano (de um curso de Engenharia). Cada sessão foi supervisionada por dois professores que permaneceram todo o tempo na sala de aula para ajudar os alunos, e observar cuidadosamente a sessão com o objetivo de obter uma opinião precisa do experimento.

Ao longo da sessão, o tempo para cada exercício e cada fase (resolução e avaliação automática com o Mooshak e visualização / animação da solução correta com o Jeliot) foi estritamente controlado, a fim de garantir que todas as subtarefas pudessem ser executadas durante a aula (2 horas). A primeira meia hora foi usada para preparar os estudantes para a sessão e o fluxograma foi divulgado e explicado. O tempo restante, os 90 minutos, foi dividido em três

partes iguais, alocando meia hora para cada exercício, ou seja, 15 minutos para desenvolver e testar uma solução e mais 15 minutos para animar uma solução correta. No final da aula cada aluno preencheu um pequeno inquérito para recolha da opinião individual sobre a experiência. Os detalhes deste experimento estão relatados no artigo (Tavares, Henriques & Gomes, 2016)

Em relação ao primeiro objectivo, os professores presentes na sala observaram e afirmaram que ambas as sessões foram executadas com êxito; nenhum incidente foi registado, sendo todas as tarefas planeadas devidamente realizadas. Em relação ao segundo objectivo, mais uma vez, os observadores relataram que todos os estudantes estiveram completamente envolvidos em terminar as atividades propostas.

À primeira vista, e de acordo com os dados recolhidos a partir de Mooshak e resumidos na Tabela 1, pensamos que o comportamento dos estudantes realmente mudou ao longo da aula, e a sua produtividade aumentou (foram resolvendo mais facilmente os exercícios propostos). Como pode ser observado na Tabela 1, o número de respostas corretas foi aumentando. A primeira linha regista o número de submissões para cada exercício que foram completamente aceites por Mooshak, isto é, que produziram o resultado esperado para todos os valores de entrada dados. As linhas dois e três registam o número total de submissões que foram avaliadas pelo Mooshak como erradas (a saída produzida não é o esperado ou foi detetado um erro de compilação ou de execução).

É importante observar que um estudante pode submeter o exercício mais do que uma vez até obter uma solução correta. Assim, o total de submissões apresentado na quarta linha mostra toda a atividade dos alunos, a sua persistência, e o grau de dificuldade dos exercícios. As três últimas linhas mostram os detalhes sobre as submissões, a fim de refinar as conclusões que podem ser tiradas a partir da quarta linha.

	Ex. 1	Ex. 2	Ex. 3
Nº de respostas corretas	4	6	9
Nº de respostas erradas	12	11	14
Nº de erros de execução	30	37	20
Nº total de submissões	46	54	43
Média de submissões	1,6	1,9	1,5
Respostas corretas após o erro	1	2	4
Nº de respostas corretas na primeira submissão	1	4	4

Tabela 1: Resumo dos resultados do experimento

Assim, uma primeira conclusão é que os estudantes resolveram o terceiro exercício mais rápido (com menor número de submissões) e com melhores resultados (número total de submissões

aceites). É importante ressaltar que o último exercício não foi mais fácil do que os anteriores. Assim sendo, na nossa opinião estas conclusões corroboram a nossa hipótese.

Também é importante salientar que os alunos mostraram uma maior dificuldade ao resolver o segundo exercício (de acordo com o número de submissões) devido ao enunciado do problema que foi um pouco mais elaborado. Além da informação numérica apresentada na Tabela 1, os dois professores presentes na sala de aula também observaram esta evidência. Esta conclusão também suporta a hipótese de que os alunos, por vezes, apresentam grandes dificuldades para entender os enunciados dos problemas. Este comentário dos professores é corroborado pela resposta dos estudantes ao questionário apresentado conforme se mostra em (Tavares, Henriques & Gomes, 2016).

4 Plataforma para o Ensino da Programação

Seguindo a abordagem 2, apresentada na Secção 3, está a ser desenvolvido um sistema de informação baseado na Web (designado por PEP) que permitirá apoiar o professor nas aulas laboratoriais e sobretudo facultará aos alunos a possibilidade de fazerem sessões de estudo fora das aulas. Tal sistema irá permitir: (i) ao docente carregar e manter os exercícios (organizados por temas e dificuldades) a usar em cada sessão, bem como fazer o plano das sessões; (ii) ao aluno executar uma ou mais sessões, para praticar um determinado tema do curso, animando os exercícios e depois resolvendo-os e testando-os com recolha imediata de feedback. A plataforma PEP vai ainda permitir que o professor possa ver a informação sobre a forma como decorreu cada sessão de trabalho de cada aluno (data e hora, sequência de exercícios resolvidos, tempo gasto, etc.). Conforme se pode ver na Figura 1, o sistema em desenvolvimento será formado por duas grandes componentes: o Back-office (BO) e o Front-office (FO). O primeiro suporta todas as tarefas de gestão da base de dados com as questões que serão posteriormente usadas pelo segundo módulo para construir as sessões que serão apresentadas aos alunos.

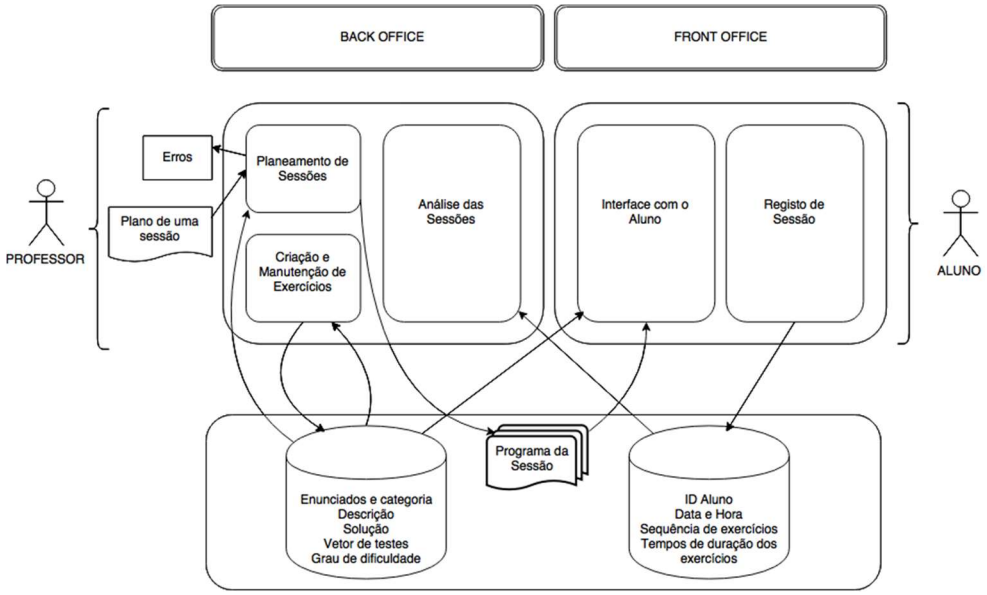

Figura 1: Arquitetura do Sistema PEP

Além disso o BO tem mais duas componentes essenciais: o compilador que lê a especificação formal de cada sessão (escrita numa linguagem de domínio específico, DSL, que criámos especificamente para este propósito) e gera o código necessário para o FO montar as sessões; e o módulo de análise que recupera da base de dados a informação relativa a cada sessão de cada aluno e a apresenta ao professor para que este possa acompanhar o processo de aprendizagem.

5 Conclusão

Neste artigo discutiu-se a necessidade de encontrar formas que melhorem o processo de ensino/aprendizagem em cursos de introdução à programação, as quais deverão atuar sobretudo na motivação e autoconfiança dos alunos que facilmente se desinteressam quando deparam com dificuldades. Nesse sentido propôs-se a combinação de duas ferramentas já existentes mas que, na nossa crença, poderão resultar mais eficazes: a animação de programas, para ajudar a compreender como funcionam as soluções aos problemas propostos; a avaliação automática de programas, para fornecer aos praticantes feedback imediato sobre as soluções por eles desenvolvidas. Foram, então, apresentadas duas abordagens que combinam de forma simétrica as duas técnicas referidas. A primeira delas já foi alvo de uma análise experimental em sala de aula, como se comenta a seguir; a segunda está a conduzir a implementação da

Plataforma para Ensino da Programação, PEP, que poderá servir para ajudar o professor na aula ou o aluno em casa.

Como já referido, em relação ao experimento realizado pode dizer-se que a evolução do comportamento dos alunos ao longo da aula de duas horas mostrou que esta abordagem levou a um melhor desempenho por parte dos mesmos. Por um lado, notou-se que o número de alunos com submissões aceites aumentou e, por outro lado, que o número de submissões aumentou e que o número de erros de compilação diminuiu. Isto significa que a motivação dos estudantes aumentou, enquanto que os erros de base foram reduzindo. A motivação foi uma das nossas principais preocupações. O experimento apresentado também nos permitiu entender a melhor forma de conduzir futuros testes, como por exemplo, permitindo uma maior flexibilidade na gestão do tempo durante a aula. Isto significa que temos a intenção de propor os três exercícios no início e permitir que os alunos escolham os intervalos de tempo para usar em cada um deles; desta forma, podem explorar mais profundamente a animação se acharem importante para sedimentar conhecimentos antes de progredir para uma nova implementação.

Para validar estas conclusões, é importante e necessário repetir o experimento para outros temas, como o processamento de vetores ou processamento de strings, envolvendo outras amostras de estudantes. Em relação ao PEP e uma vez desenhada a arquitetura e decididas as tecnologias de base de dados e Web 2.0 a usar, urge avançar com a sua implementação para o mais breve possível se proceder a testes com alunos.

Como complemento às abordagens propostas e fazendo uso dos sistemas de avaliação automática, um outro tópico que nos parece fundamental a ser estudado é o papel do trabalho em grupo na educação e o ambiente colaborativo entre estudantes conforme discutido em (Boas, Oliveira & Henriques, 2013) (Fonte, Boas, Cruz & Henriques, 2014). Neste trabalho, os autores propõem o uso de técnicas de integração contínua, habituais em Agile Development (Elliott, Fons & Randell, 2015) (Awad, 2005), para apoiar o trabalho do grupo fornecendo feedback imediato aos alunos e professores.

Agradecimentos

Este trabalho foi financiado pelo COMPETE: POCI-01-0145-FEDER-007043 e FCT – Fundação para a Ciência e Tecnologia no âmbito do projeto: UID/CEC/00319/2013.

6 Referências

ACM/IEEE (2013). Computer Science Curricula 2013 -- Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, Final Report.

Awad, M. (2005). A Comparison between Agile and Traditional Software Development Methodologies. Submitted as partial fulfilment of the requirements for the Honours Programme of the School of Computer Science and software Engineering, The University of Western Australia.

Boas, I., Oliveira, N. & Henriques, P. (2013). Agile development for education effectiveness improvement. In Proceedings of the XV international symposium on computers in education (SIIE'2013). Viseu, Portugal.

Elliott, E., Fons, F. & Randell, A. (2015). Business Architecture and Agile Methodologies. Business Architecture Guild, February.

Esteves, M. & Mendes, A. (2003). OOP-Anim, a system to support learning of basic object oriented programming concepts. International Conference on Computer Systems and Technologies - CompSysTech'2003.

Fonte, D., Boas, I., Oliveira, N., Cruz, D., Gançarski, A. & Henriques, P. (2014). Partial Correctness and Continuous Integration in Computer Supported Education. In Proceedings of the 6th International Conference on Computer Supported Education (CSEDU 2014), Volume 2. Barcelona, Spain.

Heng, P., Joy, M., Boyatt, R. & Griffiths, N. (2005). Evaluation of the BOSS Online Submission and Assessment System.

Hughes, C. & Buckley, J. (2004). Evaluating Algorithm Animation for Concurrent Systems: AComprehension-Based Approach. 16th Workshop of the Psychology of Programming Interest Group. Carlow, Ireland, April. In E. Dunican & T.R.G. Green (Eds). Proc. PPIG 16. pp. 193-205.

Hundhausen, C. & Douglas, S. (2000). Using Visualizations to Learn Algorithms: Should Students Construct Their Own, or View an Expert's? Proceedings 2000 IEEE International Symposium on Visual Languages IEEE Computer Society Press, Los Alamitos.

Joy, M., Griffiths, N. & Boyatt, R. (2005). The BOSS Online Submission and Assessment System. Journal on Educational Resources in Computing, Volume 5 Issue 3, September.

Korhonen, A. (2003). Visual Algorithm Simulation. Dissertation for the degree of Doctor of Science in Technology. At Helsinki University of Technology (Espoo, Finland).

Leal, J. & Silva, F. (2008). Using Mooshak as a Competitive Learning Tool.

Lessa, D., Czyz, J. & Jayaraman, B. (2011). JIVE: A Pedagogic Tool for Visualizing the Execution of Java Programs. SIGCSE 2011 Dallas, Texas, USA.

Mendes, A., Gomes, A. & Marcelino, M. (2004). Avaliação e Evolução de um Ambiente de Suporte à Aprendizagem da Programação. VII Congresso Iberoamericano de Informática Educativa.

Pereira, M. (2002). Systematization of Programs Animation, Sistematização da Animação de Programas. Dissertação submetida à Universidade do Minho para obtenção do grau de doutor em Informática, ramo Tecnologia da Programação.

Pereira, M., Henrriques, P. (1999). Made Algorithms Animation Systematic, Animação de Algoritmos tornada Sistemática. In 1º Workshop Computação Gráfica, Multimédia e Ensino. Leiria.

Proulx, V. (2000). Programming patterns and design patterns in the introductory computer science course. Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, pp.80-84. New York.

Queirós, R. & Leal, J. (2012). Exercises Evaluation Systems - An Interoperability Survey. In Proceedings of the 4th International Conference on Computer Supported Education (CSEDU), Volume 1, pp.83-90. Porto.

Queirós, R. & Leal, J. (2015). Ensemble: An Innovative Approach to Practice Computer Programming. In R. Queirós (Ed.), Innovative Teaching Strategies and New Learning Paradigms in Computer Programming (pp. 173-201). Hershey, PA: Information Science.

Rajala, T., Jussi, M., Erkki, L., Salakoski, K., 2007. VILLE – A Language-Independent Program Visualization Tool. Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007), Koli National Park, Finland, November 15-18.

Ramos, S. (2013). Motivação Académica dos Aluno do Ensino. http://www.psicologia.pt/artigos/ver_artigo.php?codigo=A0677 (16/03/2016)

Santos, Á., Gomes, A. & Mendes, A. (2010). Integrating New Technologies and Existing Tools to Promote Programming Learning. Algorithms, Vol3, pp.183-196.

Santos, R., & Costa, H. (2006). Análise de Motodologias e Ambientes de Ensino para Algoritmos, Estruturas de Dados e Programação aos iniciantes em Computação e Informática. INFOCOMP – JOURNAL OF COMPUTER SCIENCE, Lavras/MG – Brasil, v. 5, n. 1, p. 41-50.

Saraiya, P. (2002). Effective Features of Algorithm Visualizations. Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University for the degree of Master of Science In Computer Science, July.

Silva, M., D'Emery, R., Neto, J. & Bezerra, Y. (2009). Programming structures: A Experiment with Jeliot,Estruturas de Programação: um Experimento com Jeliot. IX Jornada de Ensino Pesquisa e Extensão (JEPEX) da UFRPE.

Tavares, P., Henriques, P. & Gomes, E. (2015). Animation and Automatic Evaluation to Support Programming Teaching. 7th International Conference on Computer Supported Education – Doctoral Consortium (CSEDU 2015). Lisboa, Portugal.

Tavares, P., Henriques, P. & Gomes, E. (2016). Computer-Supported Techniques to Increase Students Engagement in Programming. 8th International Conference on Computer Supported Education (CSEDU 2016).Rome, Italy.

Verdú, E., Regueras, L., Verdú, M., Leal, L., Castro, J. & Queirós, Q. (2011). A distributed system for learning programming on-line. Computers & Education 58, pp. 1–10.