O ENSINO DA PROGRAMAÇÃO: EXERCITAR A DISTÂNCIA PARA COMBATE ÀS DIFICULDADES

Sónia Rolland Sobral, DICT Universidade Portucalense, sonia@upt.pt

Pedro Cravo Pimenta, DSI Universidade do Minho, pimenta@dsi.uminho.pt

Resumo: As unidades curriculares de introdução à programação têm uma elevada taxa de insucesso. Os alunos têm dificuldade em esquematizar os problemas que lhes são propostos mesmo quando estes são similares a tomadas de decisão ou constituem situações normais do seu dia-a-dia. O problema coloca-se a todos os intervenientes no processo: de um lado o que ensinar e como fazê-lo; do outro lado como conseguir atingir as competências necessárias. Exercitar é a chave para conseguir melhores resultados e o ensino a distancia, sob a forma de ensino misto, poderá revelar-se uma ajuda eficaz para solucionar uma parte do problema.

Neste artigo faz-se uma reflexão sobre o ensino introdutório de programação e a forma como o ensino a distância poderá ser uma preciosa ajuda.

Palavras-chave: Ensino de programação; b-learning; estratégias de aprendizagem; Ensino Superior; interacção; ensino a distância.

1 FUNDAMENTOS E DIFICULDADES NO ENSINO DA PROGRAMAÇÃO

A ciência dos computadores*, Computer Science,* como uma ciência independente, só foi reconhecida no início da década de 60. O ensino da programação começou formalmente com a criação do departamento de informática em 1962 na Purdue University [1].

A arte de programar implica quatro passos:

a) **pensar:** fase de conceptualização e análise em que os problemas são divididos em processos ou tarefas pequenas e facilmente inteligíveis, a estruturação modular, cuja organização deve seguir uma lógica de programação descendente, ou top-down [2];

b) **resolver:** traduzir o top-down em algoritmo [3], que incorpora as regras de solução usando pseudo-código;

c) **definir:** com recurso a variáveis e estruturas de dados caracterizar o modelo dos dados a utilizar no algoritmo;

d) **formalizar**: traduzir o algoritmo para uma linguagem de programação, sua implementação e execução no computador.

Segue-se a fase mais importante, o verdadeiro momento da verdade: o programa corre, sem erros e dá o resultado correcto? E como se tem a certeza que o resultado é a ou uma solução provavelmente correcta?

Concretizando, para identificar as fases de maior dificuldade dos estudantes e imaginando um exemplo: “Sabendo o valor de um produto e respectiva taxa de IVA, determinar o preço final desse produto”.

Pensar

TOP-DOWN
Ler o valor do produto
Ler a taxa de IVA
Calcular o preço do produto
Escrever o preço do produto
Terminar

Resolver

Algoritmo Produto_com_IVA

Este algoritmo, dado o valor de um produto e a respectiva taxa de IVA, calcula e imprime o preço desse produto.

```
P10 [Ler o valor do produto]
PRINT(“Valor?”)
READ(Valor)
P20[Ler o IVA respectivo]
PRINT(“IVA?”)
READ(Iva)
P30 [Calcular o preço do produto] PP
 Valor*(1+Iva)
P40[Escrever o preço do produto]
PRINT (“O preço do produto é “,PP)
P50 [Terminar]
Exit []
```

Definir

Lista de variáveis:
Valor: Valor inicial do Produto, inteiro.
Iva: IVA do Produto, inteiro.

PP: Preço do produto com IVA, float.

Formalizar

```
// Ler o valor do produto
int
Valor=Integer.parseInt(JOptionPane.showInput
Di alog("Valor?"));  // Ler o IVA respectivo int
Iva=Integer.parseInt(JOptionPane.showInputDial
o g("IVA?"));
// Calcular o preço do produto
float PP =Valor*(1+Iva);
//Escrever o preço do produto
JOptionPane.showMessageDialog(null,"O
preço do produto é " + PP);
//Terminar
```

Por fim executar, efectuar o *run*, e verificar se o resultado obtido é o esperado.

De uma forma geral, os alunos têm dificuldade em esquematizar o problema mesmo quando ele os remete para situações relativamente usuais da sua vida diária. O problema é a compreensão, a aplicação de conceitos lógicos. No exemplo apresentado, a fase inicial é aquela que levanta mais dúvidas ao estudante. A segunda fase, resolver o problema, acaba por não ser um processo muito trabalhoso, na medida que o estudante tem "apenas" que ter conhecimento da sintaxe e características da elaboração de um algoritmo. Como os problemas tratados são de um escopo reduzido e normalmente bem conhecidos na literatura e com bibliotecas abundantes de soluções, não é difícil explicitar e formalizar as regras de negócio adequadas à solução. A terceira fase modelar os dados, não é simples. Para Niklaus Wirth [4] "um bom programa = um bom algoritmo + uma boa estrutura de dados". Exige não só um conhecimento das estruturas de dados já definidas mas também das suas condições de aplicação. No seu conceito de tipos de dados abstractos, ADT, *abstract data type*, exigem uma capacidade de abstracção para o padrão que é o ADT. Já a quarta fase de formalização é muitas vezes automatizada nos modernos ambientes de desenvolvimento, IDE, *integrated development environment*, e envolve raciocínios do tipo como, por exemplo, saber que escrever usa a palavra reservada PRINT e que as variáveis são separadas por vírgulas do texto. Para Redondo, Mendes, Marcelino, Bravo e Ortega, [5] "*contrariamente ao que muitos alunos inicialmente pensam, aprender a programar é algo mais que aprender a sintaxe de uma linguagem de programação. Essa é normalmente a parte mais fácil. A experiência diz que as maiores dificuldades observadas nos alunos residem na sua baixa capacidade de usar essa linguagens para escrever programas que resolvam problemas de forma eficaz*". Converter o algoritmo em linguagem de programação é a fase menos complicada se o aluno tiver presente qual a sintaxe da mesma.

Têm sido feitos estudos sobre a dificuldade do ensino da programação e

algoritmos [6], [7], [8]. António José Mendes [9] assinala algumas das razões para essas dificuldades, nomeadamente: elevado nível de abstracção envolvido; as metodologias tradicionais de ensino que privilegiam a aprendizagem de conceitos dinâmicos utilizando principalmente abordagens e materiais de índole estática; a necessidade de um bom nível de conhecimentos e prática de técnicas de resolução de problemas; a existência de turmas com alunos com preparação básica, *backgrounds*, diversificados acerca das matérias em questão, traduzindo-se em ritmos de aprendizagem muito diferenciado; a impossibilidade de um acompanhamento individualizado ao aluno devido à existência de turmas demasiado grandes e ao tempo dentro do currículo escolar ser demasiado apertado; a exigência de um estudo muito baseado na prática e, por isso, bastante diferente do requerido pela maioria das disciplinas (mais baseado em noções teóricas, implicando muita leitura e alguma memorização); as linguagens de programação usuais apresentam sintaxes complexas e não têm representações visuais dos algoritmos, o que não contribui para uma melhor compreensão.

Outros estudos referem causas idênticas para essa dificuldade de aprendizagem. Areias e Mendes [10] concordam com Spohrer e Soloway [11] identificando o real problema com "*colocar as peças em conjunto*" compondo e coordenando as componentes de um programa. Marcelino, Gomes, Dimitrov e Mendes [12] referem que o problema começa geralmente na fase inicial da aprendizagem: perceber e aplicar conceitos abstractos de programação, como estruturas de controlo e criar problemas para resolver problemas concretos. Marcelino, Tosheva e

Dobrodzhakiev [13] referem ainda que a situação da dificuldade de aprendizagem de disciplinas introdutórias de programação pode levar a uma baixa de confiança, menor auto-motivação e até ao abandono dos alunos. Mendes [9] diz que "claramente, a grande dificuldade para muitos alunos é que, mesmo dizendo compreender algoritmos semelhantes, se revelam

incapazes de conceber soluções para novos problemas".

A discussão sobre escolha da primeira linguagem a usar continuará sempre. É como que uma questão religiosa: uns defendem o seu Pascal por ser a melhor para ajudar os alunos a pensar, outros não vêm outra que não o Java (pela possibilidade de criar aplicações para a Internet e pelo seu paradigma ser o orientado a objectos); há quem use o paradigma lógico e hoje em dia está muito na moda a utilização do paradigma funcional...

Será a linguagem um factor tão importante no ensino da programação? Pedro Guerreiro [14] afirmava que mudar a linguagem mantendo a essência do curso, é uma atitude errada, por passar ao lado de questões fundamentais que não tem existência fora da linguagem, como passagem de parâmetros, avaliação de expressões lógicas que, caso não se esteja a pensar nalguma linguagem, os alunos adoptarão maus hábitos e subvalorizarão pontos fundamentais de programação. O debate sobre qual a linguagem e paradigma a usar continua: há quem não esteja aberto a debates e continue com paradigma imperativo numa primeira abordagem, voltando ao Pascal, usando OO, *object oriented*, numa segunda disciplina [15]. Também há quem continue a pensar que a programação estruturada é uma invenção da Universidade (Dijkstra, Wirth, Hoare, Knuth....) e que a programação por objectos é uma invenção da indústria (Booch, Stroustrup, Meyer, Coad, Rumbaugh...). Guerreiro [16], pergunta "*será que devemos usar Pascal, para aprender algoritmos? Smalltalk, para perceber as classes? O C, por causa dos apontadores? C++, quando for a sério? Eiffel, porque pode ser que venha a dar?*".

As disciplinas de algoritmia e introdução à programação têm, tradicionalmente, altíssimas taxas de insucesso, que se situam facilmente em zonas acima dos 80%, tal como acontece com outras matérias pertencentes ao domínio das ciências exactas e das Engenharias. Para os docentes estas disciplinas são aliciantes: todos os anos se tem que repensar os conteúdos, as linguagens, os paradigmas... o que motiva um professor a estar sempre a par das "modas". Isabel Alarcão [17] traduz o sentimento generalizado em "*como professora universitária fui assistindo à crescente evolução do insucesso e verificando como se intensificavam os sentimentos de frustração nos estudantes e nos professores, na grande maioria dos casos por nem uns nem outros conseguirem compreender as verdadeiras razões do que se estava a passar*".

Callear [18] sugere que os professores tenham em consideração os seguintes pontos: devem ensinar a um ritmo geralmente lento para dar tempo à assimilação de novos conceitos; um tópico de cada vez já que introduzir vários tópicos ao mesmo tempo confunde os alunos; voltar aos tópicos anteriores usando exercícios; ter atenção à ordem dos temas e os temas mais simples, que servem de base para outros, devem ser ensinados primeiro; os tópicos mais simples devem ser dados no início para dar confiança aos alunos.

Os alunos nem sempre encontram motivação para estas matérias. Nesta atitude influi muito a forma como é vista, regra geral, a profissão de programador pelos alunos de Licenciaturas em Informática. Estes sonham com profissões com nomes mais sonantes e menos conotadas com o "operário dos computadores": consultores, analistas de sistemas, directores de centros informáticos... mas não programador. Les Pinter [19] sintetiza a questão: "*Há alguns anos, quando decidi que não queria ser economista, fui trabalhar como programador para uma empresa. Percebi logo que o título "programador" não tinha propriamente uma boa conotação. Naquele tempo, os programadores estavam no nível mais baixo da pirâmide. Eles eram, ostensivamente, chamados codificadores. Se fosses competente, tornavas-te um analista de sistemas, e os programadores trabalhavam para ti e faziam aquilo que lhes dissesses. Era como que uma distinção entre oficiais e os recrutas. Se não fosses um analista de sistemas, havia alguma coisa de errado contigo. Ser um programador não tinha Status um lugar como analista de sistemas é bem mais apetecível que um emprego como programador*".

As motivações dos alunos são as mais diversas. Uma das mais correntes é a ideia de haver muita necessidade de informáticos e que esta ainda é uma profissão "de futuro" com a qual podem ter muito sucesso e ganhar muito dinheiro independentemente de terem apetência pela área ou não... No ano de 2002/2003 encontravam-se em funcionamento em Portugal 116 cursos da área de informática, sendo que em 2007/2008 havia 107 cursos.

Para ser um bom programador tem que se exercitar muito e ser bem supervisionado para conseguir um correcto estilo de programação.

2. Como conseguir obter melhores resultados

Na opinião de Areias e Mendes [10] os alunos conseguem melhores resultados tendo uma participação activa oposta ao comportamento passivo das aulas tradicionais. Ou seja, a actividade mais importante para aprender a programar é criar os seus próprios programas.

A motivação do aluno deve ser encarada como um factor essencial. A forma como a disciplina é inicialmente apresentada pode fazer a diferença. Colaço Rodrigues Júnior [20] diz que desde o primeiro dia de aulas, o professor deve explicar a sua importância dizendo que "os *fundamentos aprendidos na disciplina serão utilizados em toda vida académica e profissional*", que ao estar numa Universidade "*o aluno deve ter a consciência de que vai fazer a diferença num mercado de trabalho repleto de pseudoprofissionais de informática*". O professor deve desmistificar e desvalorizar as taxas de reprovação altas e problemas em conseguir ter sucesso dizendo que as "*dificuldades tão propaladas para a disciplina não existem, exigindo apenas uma maior atenção e resolução maciça de exercícios*". O mesmo autor fala ainda que o recheio das aulas pode constituir um verdadeiro processo de motivação se forem usadas " *perguntas e pequenos prémios para as respostas certas. Estes prémios podem ser palavras de apoio, incentivo, gratificação ou até mesmo prémios concretos ligados à área em questão, de forma que o aluno chega a estudar um pouco mais para garantir estes prémios na próxima aula*". E claro, a todos este deve sempre lembrar que aprender a programar é como fazer ginástica: um pouco todos os dias faz um bem enorme mas tentar fazer tudo de atacado pode, tal como o exercício físico a quem habitualmente não o pratica, ser altamente contraproducente.

Ao longo do tempo fizeram-se várias tentativas para melhorar o sucesso do Ensino e Aprendizagem da introdução à programação que podemos subdividir em "Pequenos Mundos" (como Alice [21] e o Karel Robot[22]); ambientes de desenvolvimento controlado (como Blue-J [23], DrScheme [24]); ferramentas de animação e simulação (SICAS [12], JAWAA[25], Jeliot [26]) e

Ferramentas específicas de linguagens (como CTutor [27] ou o MiniJava [28]). Há também a classificação feita no painel da SIGCSE'06, Tools for Teaching Introductory Programmning: What works? [29], como ferramentas narrativas (Alice [21] e Jeroo [30]), ferramentas de programação visuais (JPie [31], Karel [22]), ferramentas de modelos de fluxo (Lego Mindstorms [32]), ferramentas de linguagem (como o ProfessorJ [33]). Ou como a classificação feita por Marcelino,
Tosheva e Dobrodzhakiev [13]): mini-linguagens (MiniJava [28]), ambientes controlados de desenvolvimento (Blue-J [23]), micro-mundos (Karel[22]), ferramentas para testar soluções (WebToTeach [34]), ferramentas para simulação ou animação de algoritmos (Jeliot [26]) e programas e cursos on-line, e-livros e sites, *sítios Web*, dedicados (SICAS-W[35]).

Mendes [9] conclui que para uma ferramenta ser mais que um trabalho académico e se tornar utilizável deve ser simples, óbvia e intuitiva, portável e económica.

Verifica-se pelo simples enunciar dum rol tão extenso e meramente exemplificativo de tentativas de solução que na realidade continuamos a procurar o santo graal do mais efectivo ensino da programação. Este foi mais um incentivo para nos lançarmos nesta investigação.

3. Ensino da programação e *e-learning*

O ensino da programação ao ser complementado com o *e-learning*, na sua versão b-learning, poderia revelar-se um caminho óbvio pelo cruzamento das necessidades do primeiro e as forças do segundo.

Por outro lado é bem sabido quer das teorias de aprendizagem quer do desenho de instrução, ID, que a **motivação** é encarada como um factor essencial na procura de bons resultados [36]. A forma como a unidade é encarada sem dramatismos, encorajando os alunos a usarem a plataforma para conseguirem atingir os objectivos, pode ser dinamizada na plataforma. O reforço positivo nas respostas, a interacção rápida, o factor Web, os "prémios" para o bom trabalho, o poder trabalhar em qualquer sítio a qualquer hora, são uma mais-valia para um complemento de uma disciplina introdutória deste tipo. Também representa um modo de aplicar o modelo ARCS, a*ttention, relevance, confidence and satisfaction*, de John Keller [37] com o objectivo de promover a motivação dos alunos através dos métodos da atenção, relevância, confiança e satisfação.

Sendo o objectivo actual da educação preparar os jovens para as competências exigidas pela sociedade de informação e conhecimento

(trabalho em equipa, saber seleccionar, pesquisar, relacionar entre si e sintetizar informação, espírito criativo e capacidade de iniciativa na resolução de problemas) o construtivismo apresenta-se, mormente com Bolonha, como a teoria de aprendizagem a aplicar mais adequada aos objectivos gerais do ensino da programação. Em particular o **trabalho colaborativo** que deve estar no centro da definição de *e-learning* é um dos factores que permite que o aluno aprenda e seja obrigado a pensar.

Ainda na perspectiva construtivista **criar os seus próprios programas** e receber o retorno do docente é, como revelam Settle e Settle [38], fazer uso da interacção com o instrutor e a estrutura do curso serem os factores apontados pelos alunos para uma maior satisfação em cursos JAVA. Neste ponto o papel do docente é primordial já que o sucesso da experiência pode ser o resultado da sua actuação e participação. Como em qualquer actividade educacional construtivista o suporte, *scaffolding*, é fundamental e cabe ao professor ou instrutor fornecê-lo. Simultaneamente a interacção com o docente vai permitir que o estilo de programação que o aluno adopta seja revisto, corrigido e comentado, e levado para as boas práticas.

As características de um LMS além de satisfazerem bem, por opção de construção, as exigências de um ambiente de aprendizagem construtivista, CLE, *constructivist learning environments*, essencial no ensino da programação, encaixam nas exigências de Mendes [9] apresentadas no ponto anterior deste capítulo: **ser portável e económica, ser óbvia e intuitiva e ser simples**.

Em termos de fraquezas do ensino a distância mediado pela Web há que tomar em conta que devem ser evitadas ferramentas que exigem computadores com características específicas (apenas uma ligação à Internet, um navegador, *browser* e um leitor de ficheiros com extensão PDF).

A responsabilização do aluno pode ser tentada com a motivação e o apelo ao seu empenho. A revisão dos modelos pedagógicos deve ser alcançada, sempre tendo presente que a matéria em questão deve ser experimentada pelos alunos, “aprender fazendo”. Compete ao docente dinamizar a comunidade mostrando as vantagens do processo sendo que as expectativas elevadas estão dependentes do trabalho individual e colectivo de cada um dos intervenientes.

Por último mas não finalmente com a integração em b-learning de uma disciplina, o tempo e o volume de trabalho do docente aumentam em grande escala. Por isso o docente tem que estar consciente dessa alteração e ter sempre presente que funciona como elemento imprescindível de todo o processo de Ensino e Aprendizagem.

Bibliografia

[1] Shallit, Jeffrey. A Very Brief History of Computer Science. [Online] 1995. [Cited: 02 20, 2008.] http://www.cs.uwaterloo.ca/~shallit/Courses/134/history.html.

[2] Reis-Lima, Jorge. *Programação de computadores.* Porto : Porto Editora, 1991. 972-0-43460-0.

[3] Knuth, Donald. *The Art of computer programming.* s.l. : Addison-wesley, 1973. Vol. Volume 1: Fundamental Algorithms. 0-201-03801-3 .

[4] Wirth, Niklaus. *Algorithms + Data Structures = Programs .* New Jersey : Prentice Hall, 1975. ISBN: 0-13-022418-9.

[5] Redondo, Manuel, et al. Planificación colaborativa del diseño para el aprendizaje de la programación. Santiago, Chile : s.n., 2003.

[6] Johnson, David C. Algorithmics and programming in the school mathematics curriculum: support is waning - is there still a case to be made? [ed.] Springer Netherlands. 2000, pp. 201-214. [7] Ala-Mutka, Kirsti. PROBLEMS IN LEARNING AND TEACHING PROGRAMMING. *http://www.cs.tut.fi/~codewitz/literature_study.pdf.* [Online] 2004.

[8] Milne, I. and Rowe, G. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. *Education and Information Technologies.* 2002, pp. 55-66.

[9] Mendes, António J. *Software Educativo para Apoio à Aprendizagem de Programação.* 2002. http://www.c5.cl/ieinvestiga/actas/tise01/pags/charlas/charla_mendes.htm.

[10] Areias, Cristiana and Mendes, António. *A tool to help students to develop programming skills.* [ed.]

the 2007 international conference on Computer systems and technologies. New Yotk : ACM, 2007.

[11] Spohrer, James C. and Soloway, Elliot. Putting It All Together is Hard For Novice Programmers. Tucson, Arizona : s.n., 1985.

[12] Marcelino, Maria, et al. Using a computer-based interactive system for the development of basic algorithmic and programming skills. Bulgaria : s.n., 2004, pp. 1-6.

[13] Marcelino, Maria, et al. A Web-based approach to support initial algorithmic procedural programming learning. Coimbra : s.n., 2006.

[14] Guerreiro, Pedro. A mesma velha questão: como ensinar Programação? Mexico City : UNAM, 1986.

[15] Becker, Katrin. Back to Pascal: retro but not backwards. *Journal of Computing Sciences in Colleges.* 2002, Vols. Volume 18 , Issue 2 .

[16] Guerreiro, Pedro. The anguish of the programming teacher on the verge of becoming object-oriented. Santa Barbara, California : s.n., 1993.

[17] Alarcão, Isabel. Para uma conceptualização dos fenómenos de insucesso/sucesso escolares no ensino superior. [book auth.] José Tavares and Rui Santiago. *Ensino Superior: (in)sucesso académico.* Porto : Porto Editora, 2000.

[18] Callear, David. Teaching Programming: Some Lessons from Prolog. Heriot Watt : s.n., 2000.

[19] Pinter, Les. What's in a Name? *FoxTalk.* 1995.

[20] Colaço-Rodrigues-Júnior, Methanias. Experiências Positivas para o Ensino de Algoritmos. [Online] 2004. [Cited: 02 25, 2008.] http://www.uefs.br/erbase2004/documentos/weibase/Weibase2004Artigo001.pdf.

[21] Carnegie-Mellon, University. Alice. *Alice.* [Online] Carnegie Mellon University, 05 01, 2003. [Cited: 02 22, 2008.] http://www.alice.org.

[22] PACE-University, Seidenberg School of Computer Science and Information Systems. Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java. *Karel J. Robot.* [Online] 02 09, 2008. [Cited: 02 22, 2008.] http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html.

[23] Sun, Microsystems. Blue-J. *Blue-J - Teaching Java - Learning Java.* [Online] 01 10, 2008. [Cited: 02 22, 2008.] http://www.bluej.org/.

[24] PLanetT. DrScheme. *DrScheme.* [Online] 2008. http://www.drscheme.org/.

[25] Duke-University, Computer Science Education group. The JAWAA Homepage. [Online] 2002. [Cited: 02 22, 2008.] http://www.cs.duke.edu/csed/jawaa2/. [26] University-of-Joensuu, Department of Computer Science and Statistics. Jeliot 3. *Jeliot Home.* [Online] 10 24, 2007. [Cited: 02 25, 2008.] http://cs.joensuu.fi/jeliot/description.php.

[27] CPlusPlus. C++ Language Tutorial. *CPlusPlus.* [Online] 08 06, 2006. [Cited: 02 25, 2008.] http://www.cplusplus.com/doc/tutorial/.

[28] Cambridge-University. The Mini Java project. *The Mini Java project.* [Online] 2002. [Cited: 02 25, 2008.] http://www.cambridge.org/us/features/052182060X/index.html.

[29] Powers, Kris, et al. Tools for Teaching Introductory Programmning: What works? *SIGCSE*. Houston, Texas, USA : ACM, 2006.

[30] Georgia-Tech, College of Computing. Jeroo. *JEroo.* [Online] 07 2006. [Cited: 02 25, 2008.] http://www.cc.gatech.edu/~dorn/JerooWeb/.

[31] Washington-University, DCSE. The JPie Project. *JPie - Programming is Easy.* [Online] 2008. [Cited: 02 25, 2008.] http://jpie.cse.wustl.edu/.

[32] LEGO. LEGO MindStorms. *LEGO.com MINDSTORMS NXT Home.* [Online] 2008. [Cited: 02 25, 2008.] http://mindstorms.lego.com/.

[33] UTAH, Universty. ProfessorJ. *ProfessorJ: A teaching environment for Java with language levels.* [Online] 2008. [Cited: 02 25, 2008.] http://www.cs.utah.edu/~kathyg/profj/.

[34] Arnow, D. and Barshay, O. WebToTeach: an interactive focused programming exercise system.

Frontiers in Education Conference. 1999.

[35] Rebelo, B. Marcelino, M. and Mendes, A. Evaluation and Utilization of SICAS - A System to Support Algorithm Learning. *Proceedings of the 8th IASTED International Conference on Computers and Advanced Technology in Education CATE 2005.* Aruba : s.n., 2005.

[36] Keller, John M. Motivational design of instruction. *Instructional design theories and models: An overview of their current status.* 1983.

[37] —. *Applying the ARCS model of motivation design in Distance Learning.* 1999.

[38] Settle, Amber and Settle, Chad. Distance Learning and Strudent Satisfaction in Java Programming Courses. *Journal of Universal Computer Science.* 2007