Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte

Anabela Gomes[1], Cristiana Areias[2], Joana Henriques[3] & António José Mendes[4]

São várias as razões que tornam a aprendizagem de programação um processo difícil, ao qual as abordagens de ensino tradicionais não têm conseguido responder eficazmente. Muitas soluções tecnológicas têm sido desenvolvidas, porém o problema subsiste. Para além de diversas razões apontadas por muitos autores como estando na origem deste problema, destacamos a elevada dificuldade apresentada pelos alunos para resolver problemas genéricos. Esta dificuldade é acentuada quando os problemas são mais orientados à programação, traduzindo-se na incapacidade de concepção de algoritmos. Encontra-se em desenvolvimento uma nova proposta que se centra essencialmente no desenvolvimento de competências de resolução de problemas, genéricos numa fase de conhecimento inicial e orientados à programação em fases cognitivas mais avançadas. Este novo ambiente assenta em duas estruturas basilares, os estilos de aprendizagem preferenciais de cada aluno e o seu nível cognitivo. Relativamente a este último aspecto incorpora também mecanismos para treinar as funções cognitivas em défice.

Introdução

Os elevados níveis de insucesso nas disciplinas introdutórias de programação, em qualquer grau e sistema de ensino, é um tema que tem sido alvo de variadas discussões e pesquisas, ao longo dos tempos, resultando também em muitas propostas de metodologias e ferramentas de suporte.

1 Instituto Superior de Engenharia de Coimbra e Centro de Informática e Sistemas da Universidade de Coimbra. E-mail: anabela@isec.pt

2 Instituto Superior de Engenharia de Coimbra e Centro de Informática e Sistemas da Universidade de Coimbra. E-mail: cris@isec.pt

3 Centro de Informática e Sistemas da Universidade de Coimbra. E-mail: Joanahenriques33@hotmail.com

4 Centro de Informática e Sistemas da Universidade de Coimbra. E-mail: toze@dei.uc.pt

O ensino das linguagens de programação tem como propósito conseguir que os alunos desenvolvam um conjunto de competências necessárias para conceber programas e sistemas computacionais capazes de resolver problemas reais. Porém, a experiência tem demonstrado que existe, em termos gerais, uma grande dificuldade em compreender e aplicar certos conceitos abstractos de programação, por parte de uma percentagem significativa dos alunos que frequentam disciplinas introdutórias nesta área. Uma das grandes dificuldades reside precisamente na compreensão e, em particular, na aplicação de noções básicas, como as estruturas de controlo, à criação de algoritmos que resolvam problemas concretos. Diversos estudos apontam um conjunto de causas que estão na origem deste problema, Tobar et al. (2001), Jenkins (2002) e Lahtinen et al. (2005) são alguns exemplos.

O nosso grupo de investigação tem vindo a desenvolver alguns ambientes computacionais baseados em simulação de algoritmos e programas, tendo como objectivo facilitar a aprendizagem inicial de programação. Em colaboração com investigadores de uma universidade espanhola foi igualmente criado um ambiente colaborativo com o mesmo objectivo. No entanto, a utilização destes ambientes mostrou a necessidade de aprofundar o nosso trabalho e levar em conta aspectos não contemplados até agora, como por exemplo os estilos de aprendizagem preferenciais de cada aluno.

Neste artigo é feita uma discussão de causas para os problemas de aprendizagem de programação sentidos por muitos alunos, descrevemos brevemente algumas das ferramentas já desenvolvidas pelo nosso grupo de investigação e fazemos uma primeira descrição das preocupações e abordagens que estão a ser seguidas no nosso trabalho actual.

Causas para os problemas de aprendizagem de programação

Aprender a programar é complexo e requer esforço, perseverança e uma abordagem especial no que concerne à forma de estudo e de ensino. Há um conjunto de habilidades envolvidas que vão muito além de saber a sintaxe da linguagem. A experiência mostra que o problema começa, para muitos estudantes, numa fase inicial da aprendizagem, quando têm de compreender e aplicar certos conceitos abstractos de programação, como as estruturas do controlo, para criar algoritmos que resolvam problemas concretos. Esta é uma fase à qual é necessário prestar particular atenção, não apenas no desenvolvimento de habilidades específicas de

programação, mas também (e talvez sobretudo) na melhoria e/ou na consolidação de competências que deveriam ter sido adquiridas em anos precedentes, nomeadamente as capacidades de resolução de problemas, de raciocínio, de lógica entre outras. Porém, existe, na nossa perspectiva, um conjunto de factores mais vastos que poderão interferir neste processo, os quais passamos a abordar.

I. Métodos de ensino

Os métodos de ensino tradicionalmente utilizados não parecem adequados às necessidades da maioria dos alunos, por diferentes razões:

- **O ensino não é personalizado**. As restrições temporais levam a que seja muito difícil fornecer, em sala de aula, um *feedback* e supervisão adequados e personalizados às necessidades de cada aluno.

- **As estratégias dos professores não contemplam, habitualmente, todos os estilos de aprendizagem dos alunos.** Os indivíduos aprendem de diversas formas, tendo consequentemente preferências diversas para receber e processar novos assuntos. De acordo com os métodos tradicionais todos os alunos são forçados a uma aprendizagem uniforme, devendo aprender ao mesmo ritmo e de acordo com as estratégias pedagógicas do professor. É importante que o professor consiga contemplar a enorme diversidade de estilos de aprendizagem presentes em sala de aula. É igualmente desejável que o professor se assegure que os alunos adoptam a estratégia mais apropriada para cada assunto. Porém, o elevado número de alunos habitual em sala de aula conjugado com as limitações de tempo impossibilitam, frequentemente, abordagens mais personalizadas.

- **O ensino de conceitos dinâmicos é, normalmente, realizado através de materiais de natureza estática.** Programar envolve diversos conceitos dinâmicos que muitas vezes são ensinados através de meios estáticos (apresentações projectadas, explicações verbais, diagramas, desenhos no quadro, textos, e assim por diante) que não promovem uma plena compreensão da dinâmica envolvida.

- **Os professores estão, normalmente, mais concentrados em ensinar uma linguagem de programação e os seus detalhes sintácticos, do que em promover a resolução de problemas usando uma linguagem de programação.** A finalidade de uma disciplina de programação deve ser a de promover nos estudantes a capacidade de resolução de problemas, servindo a linguagem específica de programação apenas como uma ferramenta para expressar o algoritmo ou estratégia de

resolução. Porém, a maioria das disciplinas introdutórias de programação estão estruturadas em sentido contrário, pois é enfatizado o ensino da enorme variedade sintáctica antes de os alunos terem um bom domínio de importantes conceitos de programação.

II. Métodos de estudo

Consideramos que os métodos de estudo seguidos por muitos alunos não são apropriados para a aprendizagem de programação, essencialmente porque:

- **A programação exige um estudo muito prático e intensivo.** Muitas disciplinas, exigem metodologias de estudo à base de leituras e memorização de fórmulas ou procedimentos. Programar requer um método diferente de estudo que envolve muita compreensão, reflexão e treino. A única forma de aprender a programar é programando.

- **Os alunos não estudam o suficiente para adquirir competências de programação.** Assistir às aulas e estudar um livro de texto não é o suficiente. Programar exige um intenso trabalho de treino e reflexão extra aulas.

III. Habilidades e atitudes dos alunos

- **A maioria dos alunos apresenta enormes dificuldades em resolver problemas.** Pensamos que a causa mais importante na origem das dificuldades de programação reside num défice de capacidades de resolução de problemas genéricos. As grandes dificuldades que os alunos sentem ao programar prendem-se com a incapacidade de conceber algoritmos, e estamos convictos que estas se devem principalmente à incapacidade de resolver problemas. A resolução de problemas requer múltiplas competências que os estudantes frequentemente não têm, nomeadamente:

i) Compreensão do problema - muitas vezes os alunos começam a resolver um problema sem o compreender completamente. O que pode acontecer por dificuldades de interpretação ou por os alunos se sentirem, frequentemente, demasiado ansiosos para começar a codificar uma solução.

ii) Relacionamento de conhecimento - muitas vezes, ao resolver um problema, os alunos não estabelecem analogias com problemas seus conhecidos, perdendo a oportunidade de transferir conhecimentos prévios para novos problemas. Outras

vezes fazem-no mas tendem a agrupar problemas que têm as mesmas características superficiais e não os mesmos princípios, baseando as suas soluções em problemas não relacionados.

iii) Reflexão sobre o problema e solução - os alunos têm tendência para escrever as respostas antes de pensar cuidadosamente sobre elas. Também, no caso concreto de programação, não fazem o teste/simulação da solução construída ou quando muito fazem-no muito superficialmente ou apenas para um conjunto reduzido de testes sem verificação de casos limite.

iv) Falta de persistência - os alunos tendem a desistir de problemas cujas soluções não conseguem encontrar de forma simples e rápida. Resolver problemas de programação exige esforço e persistência. No entanto, ao enfrentar as dificuldades inerentes a um destes problemas, muitos alunos desistem passado algum tempo, preferindo pedir a solução a um colega ou simplesmente desistir. É preciso desenvolver nos alunos a consciência que o conhecimento alcançado na tentativa de resolver um problema difícil, mesmo que a solução seja imperfeita ou não conseguida, pode desenvolver estruturas cognitivas mais significativas do que conseguir resolver alguns problemas mais simples.

- **Muitos alunos apresentam défices de conhecimentos matemáticos e lógicos.** Gomes et al. (2006) conduziram algumas experiências que exploram algumas correlações entre os conhecimentos matemáticos e a falta de competências de programação. Nesta experiência os autores concluíram que os alunos envolvidos tinham dificuldades profundas em diversas áreas, tais como cálculo básico e teoria de números ou conceitos geométricos e trigonométricos simples. Os autores relatam também dificuldades relacionas com a transformação de um problema textual numa fórmula matemática que o resolva. Limitações ao nível da abstracção e do raciocínio lógico foram também identificadas.

- **Aos alunos falta "perícia" específica de programação.** Muitas dificuldades de programação são também causadas por erros específicos ou percepções erradas sobre determinados conceitos de programação. É impossível um aluno saber programar sem ter um domínio completo sobre o funcionamento de estruturas básicas de programação. É também comum que os estudantes demonstrem dificuldades em detectar erros sintácticos e lógicos de programação.

IV. Natureza da programação

- A programação envolve um elevado nível de abstracção. A aprendizagem de programação requer uma hierarquia de competências de que se destacam um elevado nível de abstracção, generalização, transferência e pensamento crítico. A experiência mostra que um dos problemas surge, numa fase de aprendizagem inicial, quando se espera que os alunos apliquem determinados conceitos abstractos de programação, como estruturas do controle, para resolver problemas reais.

- As linguagens de programação têm uma sintaxe muito complexa. A maioria das linguagens de programação foi desenvolvida com um propósito de utilização profissional e não para suportar a sua aprendizagem, sendo extensivas e com muitos detalhes sintácticos complexos e características específicas a memorizar. Essa complexidade requer que os estudantes tenham de se concentrar simultaneamente na tarefa de construção do algoritmo e na tarefa de recordar regras sintácticas.

V. Aspectos Psicológicos

- Falta de motivação. Muitos alunos não têm motivação suficiente para estudar programação, devido à conotação extremamente negativa que lhe está associada, passada de aluno em aluno. Há a imagem pública de um "programador" como um " inadequado social" (Jenkins, 2002). Adicionalmente, as disciplinas de programação adquirem a reputação de serem difíceis. Desta forma, é difícil imaginar alunos que aspirem a esta imagem, motivados para um curso difícil e com uma imagem negativa daqueles que dominam o assunto. E os estudantes que não têm motivação intrínseca, dificilmente serão bem sucedidos (Bereiter & Ng., 1991).

- Os alunos têm que aprender a programar num período difícil de sua vida. A programação é ensinada, normalmente, como um assunto básico no início de um curso superior de informática, coincidindo com um período da transição e instabilidade na vida do aluno. Alguns autores consideram que as disciplinas de programação estão mal localizadas no currículo, porque este é um momento de muitas dificuldades e novidades de uma vida nova e autónoma. O tipo de assunto é já suficientemente difícil quando os alunos estão estáveis, quando colocado num período da transição pode contribuir para aumentar ainda mais esta dificuldade (Jenkins, 2002).

Propostas de solução

Diversos tipos de ambientes informáticos, recorrendo a diversas técnicas, têm sido desenvolvidos ao longo dos últimos anos com o intuito de minorar os problemas com que os alunos se deparam na sua aprendizagem de programação. Uns recorrem a sistemas de animação com o propósito de apelar ao potencial do sistema visual humano na esperança que o formato gráfico animado contribua para uma melhor compreensão de conceitos inerentemente dinâmicos, quando comparado com o formato textual. Neste contexto surgiram diversos sistemas que recorrem a **representações visuais/animações de algoritmos.** BALSAII (Brown, 1988), Xtango (Stasko, 1992), MRUDS (Hanciles et al., 1997) Jeliot (Levy, 2003), Trackla2 (Korhonen, 2003), BlueJ (Kolling, 2003), Jhavé (Naps, 2005) são exemplos bem conhecidos.

Baseados na crença que existem vantagens cognitivas em utilizar metodologias gráficas em vez de, ou como complemento das textuais (Cilliers et al., 2005), outras propostas, para ensinar programação, desenvolveram-se no sentido da criação de **linguagens de programação baseadas em ícones**, de que se destacam o BACII (Calloni & Bagert, 1993) e de *design languages* de que é exemplo o G2 (Ellis & Lund, 1994).

Para apoiar o ensino da programação foram também desenvolvidos variados sistemas recorrendo a diversas técnicas de inteligência artificial, nomeadamente os **Sistemas de Tutores Inteligentes**, de que são exemplo, o *Lisp-Tutor* (Anderson & Reiser, 1985) e o *C-Tutor* (Song et al., 1997).

Outra proposta nesta área são os **micromundos**, seriamente influenciados pelos gráficos da tartaruga do LOGO (Papert, 1980), como por exemplo, "Karel the Robot" (Pattis, 1994), *Tortoise* (Brusilovsky, 1993) e *TurtleGraph* (Jehng et. al, 1994), Alice (Cooper et. al, 2000) e JKarel Robot (Buck & Stucki, 2001).

Mas com um conjunto tão vasto de alternativas, que segundo os seus autores e de acordo com diversos estudos efectuados, representam um contributo tão valioso para o ensino/aprendizagem de programação, porque razão é que os problemas subsistem? Porque é que essas ferramentas não são amplamente utilizadas hoje em dia? Na realidade, e pelos estudos já efectuados, parece que nenhuma das ferramentas disponíveis supre completamente as exigências da aprendizagem de programação, em especial a incapacidade de os alunos resolverem problemas, apresentarem soluções, ou seja, construírem algoritmos.

Nas próximas secções é apresentado um conjunto de ferramentas, por nós desenvolvidas, que apesar de não resolverem completamente o problema tentam combatê-lo em diversas frentes.

SICAS

O SICAS (Sistema Interactivo para Construção de Algoritmos e sua Simulação) (Gomes & Mendes, 2001), foi concebido com a preocupação principal de fornecer um ambiente onde os alunos não apenas compreendessem as diversas fases de um algoritmo já concebido, mas que sobretudo permitisse que o aluno concebesse, testasse, experimentasse, alterasse e corrigisse os seus próprios algoritmos. Possibilita, essencialmente, dois tipos de cenários: edição/resolução de problemas e execução/simulação de resoluções previamente construídas pelo aluno. No primeiro cenário, o aluno pode construir algoritmos através de representações visuais – fluxogramas – recorrendo a simbologia gráfica que representa as principais estruturas necessárias à construção de um algoritmo. No segundo cenário, o utilizador pode simular a execução das resoluções construídas, analisando-as com o detalhe e ritmo desejado.

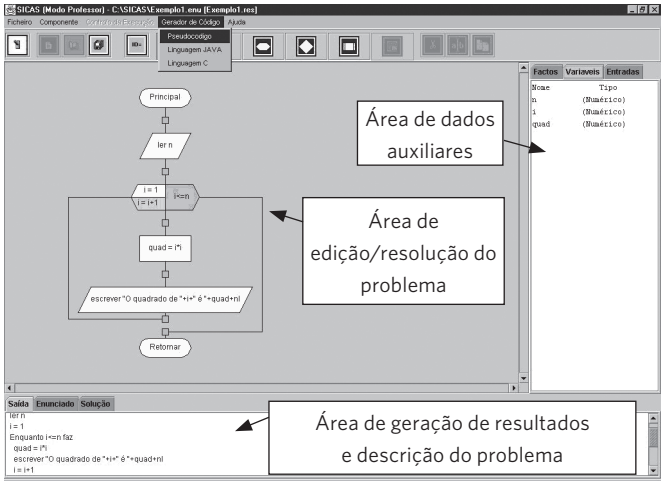

Figura 1: Construção/edição de algoritmos no SICAS

No SICAS; um problema é composto por um enunciado, uma ou mais resoluções (algoritmos) que lhe ficam associadas e, eventualmente, um conjunto de dados de teste. O ambiente de edição/resolução de problemas é constituído essencialmente

por três áreas: área de construção/edição do problema, área de dados auxiliares e área de geração de resultados e descrição do problema. Na área de edição/resolução do problema o utilizador pode construir e/ou alterar a resolução de problemas sob a forma de fluxogramas, podendo para tal recorrer a ícones ou a elementos de menus que representam as principais estruturas necessárias à sua resolução. A figura 1 ilustra este cenário.

Uma vez concluída a tarefa de construção da resolução do problema, esta poderá ser simulada pelo sistema. Para isso, o aluno terá de transitar para o modo de execução. Este ambiente permite um elevado grau de interacção e controlo por parte do utilizador, possibilitando-lhe realizar diversas configurações relativamente ao progresso de uma execução. Ao aluno é também dada a liberdade de, em qualquer momento, poder suspender a execução pelo período de tempo desejado e retomá-la quando o considerar conveniente. São também fornecidos mecanismos que auxiliam o aluno a melhor compreender os acontecimentos. Assim, de forma a melhor localizar e enquadrar o aluno relativamente à acção que está a ser simulada, o componente a ser executado em cada momento é devidamente destacado. Adicionalmente, o aluno poderá acompanhar a evolução da execução da sua resolução, conjuntamente com a inspecção das secções que fornecem informação valiosa para essa apreciação, nomeadamente as secções "Entradas", "Factos", "Saída" e "Solução". Por exemplo, ao activar o separador "Saída" o aluno poderá analisar os resultados gerados pela execução da sua resolução. Se o aluno, durante uma simulação quiser testar mais completamente a sua resolução, poderá consultar a secção "Solução". Esta, quando previamente preparada pelo professor, conterá um conjunto de resultados esperados para determinados dados de entrada ("Entradas"). A secção "Factos", possibilita ao aluno conhecer, em cada momento, os dados da simulação, estabelecendo para isso a correspondência entre cada variável utilizada na resolução e o valor que lhe está associado em cada momento. Este par variável/valor é constantemente actualizado durante a execução da resolução e é de crucial importância para a detecção de eventuais erros lógicos de concepção.

O SICAS apresenta um conjunto de possibilidades de utilização educativa que nos parecem relevantes. Destacamos a possibilidade de os alunos construírem e simularem os seus próprios algoritmos, analisando os respectivos resultados e corrigindo aspectos eventualmente menos conseguidos. Esta é uma actividade de grande importância para a aprendizagem dos fundamentos da programação, objectivo primeiro do desenvolvimento deste ambiente. Apesar de ter sido concebido para uma utilização independente, este ambiente pode também suportar

actividades em contexto de sala de aula, mesmo que o professor pretenda levar a cabo um conjunto de actividades mais dirigidas. O SICAS pode evidentemente ser utilizado em qualquer outro local, fora do horário curricular, no âmbito do estudo autónomo fundamental na aprendizagem da programação.

Outro aspecto importante da utilização do SICAS reside na possibilidade de, em utilização autónoma e sem preocupações classificativas, o aluno auto-avaliar os seus conhecimentos através da simulação e teste das suas resoluções. Em particular, a possibilidade de verificar que o seu algoritmo se comporta correctamente com os testes especificados pelo professor (dados de entrada e resultados esperados) pode apresentar uma credibilidade superior, conferindo ao aluno um grau de confiança mais elevado no sistema e nas suas próprias capacidades.

Outro aspecto importante reside na possibilidade de permitir ao professor criar conjuntos de exercícios resolvidos, constituindo assim mais um auxiliar para o estudo dos seus alunos. Claro que podem ser utilizadas diversas abordagens pedagógicas, como seja fornecer resoluções correctas, incorrectas ou ainda incompletas. Apesar de acharmos que analisar problemas resolvidos não se traduz inevitavelmente num aumento da capacidade de solucionar novos problemas, pensamos que pode ser interessante permitir aos alunos fazer alterações às resoluções existentes ou propor resoluções alternativas e testar essas novas resoluções. Também a possibilidade de dar aos alunos algoritmos errados, em especial com o tipo de erros lógicos que o aluno em causa habitualmente apresenta, e pedir-lhe que procure e corrija esses erros, pode apresentar um alto valor educativo nesta área.

Outra característica importante do SICAS é a possibilidade de os alunos compararem algoritmos diferentes para um mesmo problema e verificarem quando é que uns apresentam um desempenho superior aos outros, interiorizando assim gradualmente técnicas eficazes de programação. O ambiente pode também ser utilizado numa inversão de papéis, podendo ser pedido ao aluno para indicar o enunciado de um problema cuja resolução lhe seja facultada (e que ele pode analisar com o SICAS).

Destacamos ainda a possibilidade de o professor colocar problemas aos alunos e verificar o seu grau de proficiência através da análise das resoluções por eles realizadas. De acordo com esta perspectiva, é possível afirmar que o SICAS permite avaliar e individualizar as actividades desenvolvidas pelos alunos. Com este conhecimento, o professor pode propor actividades de acordo com os níveis actuais de conhecimentos de cada aluno, evitando propor problemas demasiado fáceis

ou difíceis, o que geralmente se traduz em desmotivação dos alunos. Este aspecto acrescenta um conjunto de valores importantíssimo aos métodos de ensino tradicionais, na medida em que possibilita uma actividade de ensino/aprendizagem mais personalizada, contribuindo para que os alunos possam aprender ao seu próprio ritmo, aumentando a sua motivação, pois muitas vezes os factores que mais contribuem para o desinteresse dos alunos nas salas de aula é a sua total incapacidade de acompanhamento dos exercícios que estão a ser abordados. É claro que este objectivo poderia ser atingido por outros meios, mas pensamos que o SICAS pode proporcionar algum suporte a esta abordagem, necessariamente mais trabalhosa para o professor.

Contudo, as avaliações efectuadas com o SICAS demonstraram que a abordagem utilizada não é suficiente para contemplar todos os alunos. Por um lado, é um ambiente que não promove de igual forma todos os estilos de aprendizagem, mas antes favorece os alunos marcadamente visuais em detrimento dos verbais. Por outro lado, constitui uma abordagem útil apenas para aqueles alunos que quando confrontados com o enunciado de um problema conseguem iniciar a sua resolução e construir uma primeira solução, mesmo que não completamente correcta. Com base na simulação dessa solução estes alunos podem verificar quais os erros cometidos e proceder à sua correcção.

No entanto, existem alunos mais fracos, que perante o enunciado de um problema nem sequer conseguem chegar a uma primeira proposta de solução. Nestes casos o SICAS é de pouca utilidade, pois não tem mecanismos que procurem levar estes alunos a ultrapassar a situação de impasse em que muitas vezes se encontram. Pensamos que estes alunos precisam, numa primeira fase, de uma orientação mais forte que os motive e ajude a estruturar ideias, de modo a que possam chegar a uma solução para o problema. O PROGUIDE, descrito na secção seguinte, pretende responder a este problema.

PROGUIDE

Quando os alunos começam a programar utilizando uma abordagem procedimental, existe um conjunto de conhecimentos básicos que têm de ser adquiridos, dos quais destacamos o conceito de variável e tipos de dados básicos, formas de comunicação com o utilizador e estruturas de controlo. Apesar destes conhecimentos serem transmitidos, e de a grande maioria dos alunos ser capaz de os entender, a verdade é que muitos deles dificilmente os conseguem aplicar quando confrontados

com um determinado exercício. Por exemplo, quando lhes é pedido para criar um algoritmo de contagem, muitos alunos não conseguem perceber a necessidade de usar um ciclo com uma variável a ser incrementada, enquanto outros, mesmo tendo essa percepção, não a conseguem operacionalizar. Outras vezes os alunos não conseguem sequer iniciar a resolução de um problema, e se a iniciam, fazem-no por um caminho errado, seja por não entenderam o que se pretende, ou por criarem modelos mentais incorrectos. Quando os alunos se deparam com a descrição textual de enunciados de problemas, encontram, muitas vezes, dificuldades em extrair as informações necessárias para iniciar a sua resolução. Outros, apesar de entenderem o enunciado e até terem uma ideia geral da solução, nem sempre conseguem descrever os passos que a constituem. Este tipo de situações faz com que desde cedo, surjam dificuldades e erros que são repercutidos para as próximas fases de aprendizagem, contribuindo para uma desmotivação e abandono deste tipo de estudo. É neste sentido que foi desenvolvido o ProGuide, um ambiente de aprendizagem essencialmente orientado para a resolução de problemas básicos, dando especial ênfase à fase de planeamento e concepção da solução, incentivando e guiando o aluno de forma a atingir uma solução correcta (Areias, 2007). Desta forma procura-se que os alunos se sintam mais acompanhados e que consigam atingir um nível básico de sucesso que os possa levar posteriormente a melhor desenvolver as suas capacidades de programação. O diálogo estabelecido entre o ProGuide e o aluno é feito num subconjunto de linguagem natural, procurando desta forma facilitar a comunicação com o aluno com dificuldades. Veja-se a figura 2.

No ProGuide, procura-se levar o aluno a responder a questões relevantes para a concepção da solução, como por exemplo:

- Quais são as variáveis conhecidas e desconhecidas do problema.
- Quais as informações que podem ser obtidas a partir do enunciado do problema.
- O que precisa ser calculado ou executado.
- Que informação deve ser obtida do utilizador.
- O que é que o programa deve apresentar como resultado.
- Quais os passos para resolver o problema.
- Que tipo de elementos devem ser incluídos no fluxograma para efectuar o que se pretende.
- Identificar se deve ser introduzido um ciclo, selecção ou elemento de entrada/saída de dados.

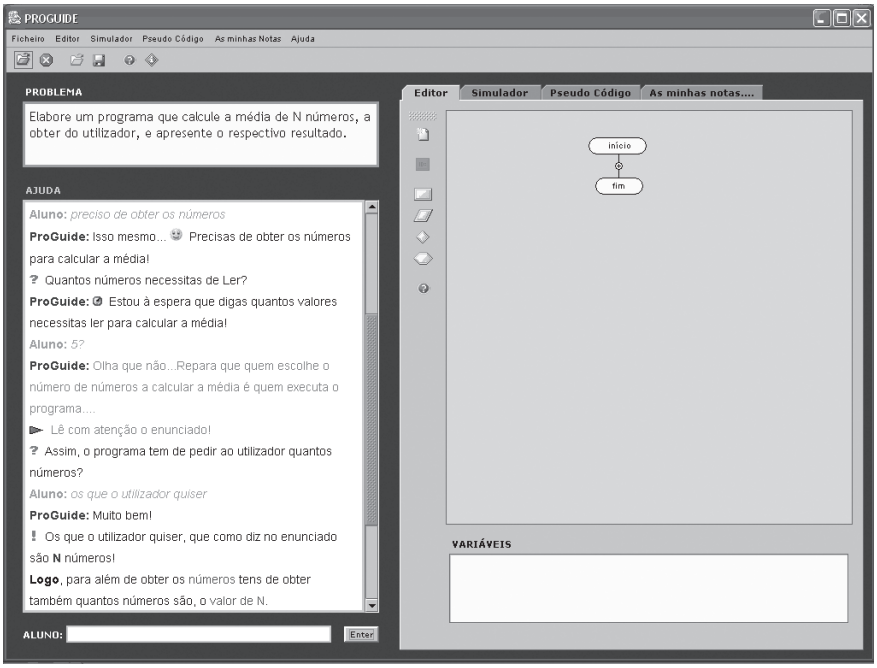

Figura 2: Simulação de um programa no PROGUIDE

De forma a avaliar a sequência do diálogo, as acções do aluno vão sendo monitorizadas, sendo possível deste modo ao ambiente intervir de forma a ajudar e motivar os alunos, reduzindo o sentimento de desamparo que muitas vezes surge quando se encontram a resolver problemas sozinhos.

O sistema permite ainda que o aluno simule as suas resoluções facilitando a análise e compreensão de erros eventualmente cometidos. Para além do suporte à criação de soluções para problemas, o sistema possibilita também ao aluno obter explicações sobre determinados assuntos indispensáveis, de forma a dar os primeiros passos em programação. Entre eles destacam-se as variáveis, a comunicação com o utilizador, os ciclos e as selecções. Para cada um destes assuntos é apresentado um conjunto de exemplos práticos e a respectiva explicação, sendo possível que o aluno os simule de forma a melhorar a sua aprendizagem.

Porém, este sistema também requer alguns melhoramentos, no que concerne ao processamento de linguagem natural que é difícil e complexo, processando actualmente apenas um subconjunto limitado da linguagem Portuguesa.

Projectos colaborativos

Uma outra vertente em que também já investimos diz respeito ao desenvolvimento de ferramentas colaborativas de suporte à resolução conjunta de problemas de programação procedimental. Assim, surgiu a ideia de ligar a ferramenta de simulação SICAS desenvolvida pela equipa da Universidade de Coimbra à ferramenta colaborativa PlanEdit (Redondo, et al., 2002) desenvolvida pelo grupo CHICO da Universidade de Castilla-La Mancha, em Espanha.

O PlanEdit é uma ferramenta, originalmente desenvolvida como um componente do DomoSim-TPC, de suporte à resolução colaborativa de problemas no campo da Domótica. Porém, foi adaptada para a área da programação, uma vez que esta é também uma actividade de resolução de problemas. Em particular, a sua ferramenta de discussão argumentativa revelou-se de grande importância aquando da sua utilização conjunta com o SICAS, para possibilitar a resolução de problemas de programação em grupo. Através desta integração surgiu o SICAS-COL (Rebelo, 2007) ou seja uma nova versão do SICAS com suporte a trabalho colaborativo.

Todas estes desenvolvimentos se justificam em diferentes níveis de aprendizagem do aluno. Assim, um aluno com fracos conhecimentos poderá começar com o PROGUIDE. Após o desenvolvimento de algumas competências que já lhe permitam iniciar e desenvolver soluções simples, poderá trabalhar individualmente utilizando o SICAS. Poderá também iniciar o trabalho em grupo recorrendo ao SICAS-COL, usado para apoiar interacções entre alunos, mesmo que à distância. As propostas de solução de cada problema continuam a ser expressas através dos fluxogramas do SICAS, mas nesta fase todos os membros do grupo as podem analisar, comentar ou criticar e, possivelmente, contribuir para a sua melhoria. Acreditamos que estas actividades de análise, reflexão e discussão colaborativa, podem melhorar a aprendizagem do estudante.

Trabalho em curso

Pensamos que utilizar apenas as ferramentas descritas não será ainda suficiente para criar um ambiente que possa suportar todos os estudantes eficientemente. Outros aspectos terão também que ser considerados. Por exemplo, os estudantes têm estilos de aprendizagem diferentes e enquanto para muitos as ferramentas anteriormente descritas podem ser suficientes, outros podem tirar pouco proveito delas por não contemplarem os seus estilos de aprendizagem preferenciais, sendo necessárias outras abordagens.

Assim, pensamos que é necessário um novo ambiente de aprendizagem que deverá detectar o estilo de aprendizagem preferencial de cada aluno, apresentando-lhe conteúdos e actividades em concordância (Gomes et al., 2007). Para a determinação do estilo de aprendizagem dos indivíduos existem diferentes modelos, nomeadamente os "The Myers-Briggs Type Indicator – MBTI - (Myers, 1985)", "The Kolb's Learning Style Model"(Kolb, 1985) e "The Felder-Silverman Learning Style Model" (Felder, 1988) entre outros. No entanto, decidiu-se seguir o modelo de "The Felder--Silverman Learning Style Model" e adoptar o tipo de inquérito nele proposto para diagnosticar os estilos de aprendizagem dos alunos. A principal razão da escolha deste modelo prende-se com o facto de ter sido desenvolvido a pensar em alunos de engenharia, também o nosso público-alvo, para além de possuir um inquérito on-line que facilmente permite caracterizar um indivíduo.

Um nível mais elevado de adaptabilidade deverá também ser alcançado pelo facto do ambiente de aprendizagem manter um registo de cada actividade do aluno e diagnosticar padrões dos seus erros, áreas fortes e dificuldades. Julga-se ser extremamente útil definir um modelo que possibilite fazer um acompanhamento individualizado sobre o estado cognitivo do estudante, em cada assunto de programação. Isto para permitir seguir uma sequência correcta de aprendizagem, que oriente o aluno para um nível de conhecimentos adequado e onde não se perca tempo com conceitos que o aluno já sabe e domina, concentrando os esforços nas reais dificuldades de cada estudante. Isto permitirá uma interacção mais personalizada com cada aluno. O professor pode também usar esta informação para melhor conhecer o nível de cada estudante (isto é geralmente difícil devido ao número elevado dos estudantes em cada curso) e adoptar estratégias diferentes para ajudar todos os alunos.

Como, na nossa opinião o grande problema se prende com a incapacidade de resolução de problemas, pretende-se colmatar uma lacuna não contemplada nos sistemas conhecidos, o desenvolvimento da capacidade de resolução de problemas genéricos orientados à programação. Assim, uma parte fundamental do novo ambiente, consistirá na incorporação de vários tipos de actividades lúdicas e jogos lógicos que, de uma forma atractiva e estimulante, permitam desenvolver a capacidade de resolução de problemas nos alunos. Numa fase de conhecimento inicial os problemas a resolver serão de diversos domínios (quebra-cabeças simbólicos, quebra-cabeças lógicos, jogos e charadas, problemas simples de aritmética e geometria, entre outros) não tratando directamente de algoritmos ou de programação. À medida que o nível de conhecimento do aluno evolui também o tipo de proble-

mas apresentado progride gradualmente orientando o aluno para a construção de algoritmos. O aspecto principal residirá em, através de um ambiente estimulante e atractivo, averiguar o progresso do aluno, nos aspectos de capacidade de abstracção, raciocínio lógico, solução de problemas e autonomia cognitiva. De notar que, cada uma das fases apontadas serão sempre aplicadas de acordo com o estado actual de conhecimento do aluno e do seu estilo preferencial de aprendizagem.

Para tal, está-se presentemente a elaborar um estudo sobre os mecanismos/modelos cognitivos que são utilizados na aprendizagem da programação. Apesar de existirem várias taxonomias relativas aos processos cognitivos implicados no desenvolvimento de tarefas gerais e de resolução de problemas, das nossas pesquisas, não foi encontrado qualquer estudo similar aplicado à programação, pelo que tencionamos construí-lo. Após essa construção pretende-se definir um conjunto de testes para avaliar cada uma das funções cognitivas definidas nessa taxonomia. Serão também construídos testes para treinar as funções cognitivas presentes nessa taxonomia, que poderão estar em défice em determinado aluno.

Pretende-se também diversificar o tipo de actividades, por exemplo, fornecendo exemplos correctos, de forma a exemplificar boas práticas de programação, exemplos incorrectos (especialmente com o tipo de erros lógicos cometidos pelos alunos), para detecção de erros, exercícios/actividades para completar, entre outros. Exemplos de actividades encontram-se descritas em (Gomes et. al., 2007)

Conclusão

O insucesso generalizado verificado nas disciplinas de programação tem constituído uma preocupação por parte de muitos investigadores e educadores. Entre vários factores referidos como causas das dificuldades de programação os autores apontam a habilidade de resolução de problemas.

Diversos sistemas têm sido propostos ao longo do tempo numa tentativa de ultrapassar ou, pelo menos, minorar o problema. Também a nossa equipa tem vindo a desenvolver diversos sistemas para atacar o problema, tendo em mente alunos com diferentes tipos de dificuldades. No entanto, o problema ainda está longe de ser resolvido.

A abordagem presentemente em desenvolvimento centra-se em actividades de resolução de problemas genéricos, encaminhando o aluno progressivamente para a programação. A grande preocupação residirá em tentar contemplar todos os estilos

de aprendizagem preferenciais dos alunos, incorporando adicionalmente ferramentas que permitam treinar os eventuais défices cognitivos. A ideia é que cada actividade proposta ao aluno esteja não apenas de acordo com o seu estilo preferencial de aprendizagem mas também de acordo com o seu estado actual de conhecimentos.

Bibliografia

Anderson, J. R. & Reiser, B. J. (1985). The LISP tutor. *Byte*, 10 (4), 159-175.

Areias, C. (2007). *ProGuide: Sistema de acompanhamento na resolução de problemas básicos de programação.* Tese de Mestrado em Engenharia Informática, Universidade de Coimbra.

Bereiter, C. & Ng., E. (1991). Three Levels of Goal Orientation in Learning. *Journal of the Learning Sciences,* 1 (3), 243-271.

Brown, M. (1988). Exploring algorithms using BALSA-II. *IEEE Computer*, 21 (5), 14-36.

Buck, D. & Stucki, D. J. (2001). JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum". *ACM SIGCSE Bulletin*, 33 (1), 16-20.

Brusilovsky, P. (1993). Program visualization as a debugging tool for novices. *Proceedings of INTERCHI'93*, Amesterdão, 24-29.

Calloni, B. A. & Bagert, D. J. (1993). BACCII: An iconic syntax-directed system for teaching procedural programming. *Proceedings of the 31st ACM Southeast Conference*, Birmingham, 177-183.

Cilliers, C., Calitz, A. & Greyling, J. (2005). The effect of integrating an iconic programming notation in CS1. *Proceedings of the 10th Annuak SIGCSE Conference on Innovation and Technology in Computer Science Education*, Monte de Caparica, Portugal, 89-93.

Cooper, S., Dann, W. & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing in Small Colleges*, 15 (5), 107-116.

Ellis, G. P. & Lund, G. R. (1994). G2- A design language to help novice C programmers. Retrieved December 9, 2007, from the World Wide Web: http://www.comp.lancs.ac.uk/~ellisg2/

Felder, R. M. (1988). Learning and Teaching Styles in Engineering Education. *Journal of Engineering Education*, 78 (7), 674-681.

Gomes, A., Carmo, L., Bigotte, E. & Mendes, A. J. (2006). Mathematics and programming problem solving. *Proceedings of 3rd E-Learning Conference – Computer Science Education* (CD-ROM), Coimbra, Portugal.

Gomes, A. & Mendes, A. J. (2001). SICAS: Interactive system for algorithm development and simulation. In M. Ortega & J. Bravo (Ed.), *Computers and Education in an Interconnected Society* (pp. 159-166). Kluwer Academic Publishers.

Gomes, A., Santos, A., Carmo, L. & Mendes, A. J. (2007). Learning styles in an e-learning tool. *Proceedings of the International Conference on Engineering Education-ICEE'07.*

Hanciles, B., Shankararaman, V. & Munoz, J. (1997). Multiple representations for understanding data structures. *Computers & Education*, 29 (1), 1-11.

Jehng, J., Shih, Y., Liang, S. & Chan, T. (1994). Turtle-Graph: A computer Supported Cooperative learning environment. *Proceedings of the ED-MEDIA'94*, 293-298.

Jenkins, T., (2002). On the difficulty of learning to program. *Proceedings of the 3rd Annual LTSN_ICS Conference*, Loughborough University, United Kingdom, 53-58.

Lahtinen, E., Ala-Mutka, K. & Järvinen (2005). A studdy of difficulties of novice programmers. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education,* Monte de Caparica, Portugal, 64-68.

Levy, R. B., Ben-Ari, M. & Uronen, P. A. (2003). The Jeliot 2000 program animation system. *Computers & Education*, 40 (1), 1–15.

Myers, I. B. & McCaulley, M.H. (1985). *Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator.* Palo Alto, CA: Consulting Psychologists Press.

Naps, T. (2005). Jhavé – Supporting Algorithm Visualization. *IEEE Computer Graphics and Applications,* 25 (5), 49-55.

Papert, S. (1980). *Mindstorms, children, computers and powerful ideias.* New York: Basic Books.

Pattis, R. (1994). *Karel the Robot: A gentle introduction to the art of programming* (2nd Edition). John Wiley & Sons.

Kolb, D. A. (1985). *Learning Style Inventory: Technical Manual.* McBer and Company, Boston.

Kolling, M., Quig, B., Patterson, A. & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computing Science Education,* Special Issue of Learning and Teaching Object Technology, 12 (4), 249–268.

Korhonen, A., Malmi, L. & Silvasti, P. (2003). TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. *Proceedings of the 3rd Finnish/Baltic Sea Conference on Computer Science Education,* Koli, Finlândia, 48-56.

Rebelo, B. (2007). *SICAS-COL - Um Sistema Colaborativo para Aprendizagem Inicial da Programação.* Tese de Mestrado em Engenharia Informática, Universidade de Coimbra.

Redondo, M., Bravo, C., Ortega, M. & Verdejo, M. (2002). PlanEdit: An adaptive tool for design learning by problem solving. *Proceedings of 2º Adaptive Hypermedia and Adaptive Web-Based Systems (AH2002). LNCS 2347,* Berlin: Springer-Verlag, 560-563.

Song, J. S., Hahn, S. H., Tak, K. Y. & Kim, J. H. (1997). An intelligent tutoring system for introductory C language course. *Computers & Education*, 28 (2), 93-102.

Stasko, J. (1992). Animating algorithms with XTANGO. *SIGACT News*, 23 (2), 67-71.

Tobar, C. M., Rosa, J.L. G., Coelho, J. M. A. & Pannain, R. (2001). Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação. *Actas do XII Simpósio Brasileiro de Informática na Educação (SBIE'2001).* Vitória, ES, Brasil.

Résumé

Il y a plusieurs des raisons qui rendent l`apprentissage de programmation un procès difficile, auquel les abordages de l`enseignement traditionnel n`ont pas obtenu une réponse efficace. Beaucoup des solutions technologiques ont été développées, toutefois le pròbleme subsiste. Outre les diverses raisons étudiées par de nombreux auteurs comme étant à l`origine de ce problème, se détache la grand difficulté présentée par les élèves dans la résolution des problèmes génériques.

Cette difficulté est accentuée quand les problèmes sont plus orientés pour la programmation, se traduisante dans l`incapacité de conception d`algorithmes simples. Il se trouve en développement une nouvelle proposition qui se centre essentiellement sur le développement des compétences de résolution de problèmes génériques, dans une phase de connaissance initiale et orientés pour la programmation, dans des phases cognitives plus avancées. Dans la base de cette nouvelle proposition nous avons deux structures basilaires: les styles de apprentissage préférentieles de chaque élève et son niveau cognitif. Relativement à ce dernier aspect il incorpore aussi des mécanismes pour entraîner les fonctions cognitives en déficit.

Abstract

Programming learning is known to be a difficult task to many students. Traditional teaching and learning strategies have failed to solve this problem to an acceptable level. Also many computer-based tools have been proposed, but the problem still remains. Many authors have studied the reasons for programming learning difficulties, but we think the lack of good generic problem solving skills is the main cause for many students problems. When problems are more programming oriented the difficulties often increase and students show big difficulties even to create simple algorithms. Our current work is centered around the development of problem solving skills, first for generic problems and later for programming problems. This new environment takes into consideration each student preferential learning style and current problem solving level.