REVISTA GALEGO-PORTUGUESA DE PSICOLOXÍA E EDUCACIÓN

Vol. 19, (nº1), Ano 16º-2011 ISSN: 1138-1663

PROGRAMAÇÃO DE COMPUTADORES: COMPREENDER AS DIFICULDADES DE APRENDIZAGEM DOS ALUNOS

COMPUTER PROGRAMMING: UNDERSTANDING THE STUDENTS' LEARNING DIFFICULTIES

***Ana Paula L. AMBRÓSIO*[1]**
***Leandro S. ALMEIDA*[2]**
***Joaquim MACEDO*[3]**
***Alexandre SANTOS*[3]**
***Amanda H. FRANCO*[2]**
Universidade Federal de Goiás, Instituto de Informática[1]
Universidade do Minho, Instituto de Educação[2]
Universidade do Minho, Escola de Engenharia[3]

Data de recepción: 04/03/2011
Data de aceptación: 27/06/2011

RESUMO:

As dificuldades dos alunos na aprendizagem das disciplinas de programação de computadores são frequentes, explicando o insucesso académico e o abandono de alguns alunos nos cursos de informática. Neste artigo, descrevemos tais dificuldades tomando o discurso de professores e de alunos, contrastando aqui alunos com bom e fraco rendimento. Neste sentido, após um levantamento da literatura internacional na área, descrevemos as percepções de professores e alunos sobre as exigências cognitivas e académicas da aprendizagem de programação. Os resultados obtidos sugerem que os fracos alunos apresentam claras dificuldades na abstracção dos problemas e tendem para uma resolução por tentativa e erro dos problemas. Os bons alunos, descrevem uma maior concentração dos seus esforços na análise aprofundada do problema, sua partição e representação mental, avançando para a resolução com maior clareza cognitiva, organizando a solução em passos sequenciais.

PALAVRAS-CHAVE: Programação de Computadores; Informática; Raciocínio algorítmico; Aptidões cognitivas; Ensino Superior.

ABSTRACT

Student's learning difficulties in university computer programming courses are frequent and often insurmountable, leading to high

Correspondencia:
E-mail: apaula@inf.ufg.br; leandro@ie.uminho.pt; amanda.franco@ie.uminho.pt; macedo@di.uminho.pt; alex@di.uminho.

failure and dropout rates. This paper describes these difficulties based on the discourse of teachers and students, comparing students with high and low performance. In this sense, after a survey of international literature in the domain, this study presents teachers' and students' perceptions about the cognitive and academic requirements of learning to program. The results suggest that low achievers present clear problem abstraction difficulties and lean towards solving problems by trial and error. Good students show greater concentration effort on a profound analysis of the problem, its partition and mental representation, advancing towards a solution with greater cognitive clarity, organizing the solution in sequential steps.

KEY-WORDS: Computer Programming, Computer Science, Algorithmic reasoning, Cognitive aptitudes, Higher Education

RESUMEN

Las dificultades de los alumnos en el aprendizaje de las disciplinas de programación de ordenadores son frecuentes, explicando el fracaso académico y el abandono de algunos alumnos de los cursos de informática. En este artículo, describimos estas dificultades tomando el discurso de maestros y de alumnos, diferenciando alumnos con un bueno y un malo rendimiento. En este sentido, después de hacer el levantamiento de la literatura internacional en el área, describimos las percepciones de maestros y alumnos acerca de las exigencias cognitivas y académicas del aprendizaje de programación. Los resultados obtenidos sugieren que los malos alumnos presentan claras dificultades en hacer la abstracción de los problemas y tienden a una resolución de los problemas por tentativa y error. Los buenos alumnos describen una mayor concentración de sus esfuerzos en el análisis profundada del problema, su partición y representación mental, avanzando para la resolución con una mayor clareza cognitiva, organizando la solución en pasos secuenciales.

PALABRAS CLAVE: Programación de ordenadores; Informática; Razonamiento algorítmico; Aptitudes cognitivas; Enseño Superior.

INTRODUÇÃO

O ensino de Programação de Computadores apresenta desafios que persistem após 50 anos de pesquisas na área. Aprender a Programar aparece entendido como uma tarefa particularmente difícil, envolvendo diversos conhecimentos e habilidades (Jenkins, 2002). As pesquisas nesta área cedo consensualizaram que a programação é uma actividade altamente complexa, que envolve sub-tarefas ligadas a diferentes domínios de conhecimento e a uma variedade de processos cognitivos (Pea & Kurland, 1984). Assim, valoriza-se um conjunto de competências: a compreensão leitora; o raciocínio crítico e o pensamento sistémico; as metcomponentes cognitivas de identificação, planeamento e resolução de problemas; a criatividade e curiosidade intelectual; as habilidades matemáticas e o raciocínio condicional; o pensamento procedimental e o raciocínio temporal; o raciocínio analítico e o raciocínio quantitativo; ou ainda, o raciocínio analógico, silogístico e combinatório. De clarificar que várias destas competências se sobrepõem, apenas reflectindo diferentes designações usadas pelos diversos autores.

No ensino superior, as unidades curriculares de programação geralmente aparecem no 1º semestre do curso, assim que os alunos ingressam na universidade. As disciplinas introdutórias de programação são referenciadas na literatura como CS1 (Computer Science 1), e geralmente estão associadas a um grande número de reprovações, de desistências na própria disciplina e, inclusive, de abandonos do curso. Estas dificuldades ampliam o seu impacto se considerarmos a fase de transição académica em que os estudantes se encontram e os processos nem sempre fáceis da sua adaptação ao ensino superior (Almeida, 2007).

Tudo isto é socialmente preocupante quando se verifica que, apesar da crescente necessidade de profissionais da área de computação, o número de inscritos e de diplomados nestes cursos tem diminuído.

Alunos ingressantes em cursos de computação apresentam comportamentos distintos nas disciplinas introdutórias relacionadas com a programação. Enquanto alguns conseguem aprender a programar rapidamente, inclusive com relativa facilidade, outros experienciam enormes dificuldades (Leither & Lewis, 1978). Destes últimos, alguns conseguem, num dado momento, "dar um salto quântico" e avançam na disciplina. Outros, no entanto, mesmo demonstrando enorme empenho, não chegam a tal "salto". Mesmo não sendo conhecidas as condições percursoras deste "salto", verifica-se que a generalidade dos alunos, ao persistir na aprendizagem de programação, acaba por consegui-lo.

No intuito de tentar identificar as condições e o momento em que este "salto" acontece, foi conduzido um estudo qualitativo com alunos e professores do Brasil e de Portugal. Neste processo, foram usadas entrevistas para identificar os factores que eles acreditam ser decisivos para a aprendizagem de computação e a fase do processo de programação em que os alunos encontram maiores dificuldades. Os resultados serão usados para definir ferramentas que possam identificar os alunos em risco e para promover estratégias de facilitação do sucesso académica na disciplina. Para permitir uma melhor compreensão do problema, e definir uma base comum de discussão, apresentamos uma descrição do que se entende por "programar", seguida de uma retrospectiva da investigação internacional na área.

PROGRAMAR: DELIMITAR A SUA APRENDIZAGEM

De forma geral, programar é vista como uma forma de resolução de problemas que pode ser decomposta em quatro fases: entender o problema, definir ou planear uma solução, implementar um plano, e verificar se está certo ou se é adequado o resultado final atingido. Pensando nestas quatro fases, antecipamos a diversidade de competências e de conhecimentos envolvidos.

O objectivo da disciplina CS1 é ensinar aos alunos os conceitos básicos de programação, permitindo que implementem soluções para problemas numa determinada linguagem de programação. Nesta disciplina, são apresentados os comandos existentes nas linguagens e que podem ser usados para escrever um programa. Nas disciplinas iniciais, isto representa um pequeno conjunto de comandos e conceitos.

Na posse deste conhecimento, os alunos devem elaborar soluções para os problemas propostos. Para isto, devem ler o enunciado do problema e entendê-lo, recorrendo, com frequência, a uma execução manual do mesmo, através da atribuição de valores para as variáveis. Este processo resolutivo pode ser feito várias vezes, chegando a uma generalização, a partir da qual é elaborado um algoritmo que descreve a solução como uma sequência de comandos passíveis de serem executados por um computador. Este algoritmo é uma representação da lógica de solução do problema. Nesta fase, os alunos preocupam-se em organizar os comandos de tal forma que, dados os valores de entrada, estes sejam manipulados para produzir uma saída correcta. Para verificar a validade e correcção do algoritmo desenvolvido, muitas vezes recorre-se a uma simulação, onde o aluno se coloca no lugar do computador, executando os passos descritos pelos comandos e tentando identificar potenciais erros.

Este algoritmo é então traduzido para uma dada linguagem de programação, tendo-se particular atenção à sintaxe. Assegurada a correção sintática do programa, verifica-se a

correcção semântica. Para isto, são definidos casos de teste onde valores de entrada devem gerar valores de saída apropriados. Caso o programa não esteja correcto, este deve ser modificado a fim de ultrapassar os erros, passando novamente pelos passos de análise, codificação e testagem.

Na prática de sala de aula observa-se que, mesmo entendendo o problema, e sendo capazes de resolver uma instância deste problema, os alunos sentem enorme dificuldade em traduzir esta solução numa sequência de comandos executáveis no computador. Isto verifica-se de uma forma clara quando se apresentam problemas do tipo "ordene um conjunto de números em ordem crescente". Os alunos entendem o enunciado do problema, no entanto encontram enormes dificuldades em definir uma sequência de comandos que, se executados pelo computador, dado um conjunto de números, leve ao resultado desejado. Estas observações são corroboradas por Winslow (1996) que conclui que "noviços conhecem a sintaxe e semântica de comandos individuais, mas não sabem como combiná-los para obter programas válidos. Mesmo quando sabem resolver os problemas à mão, têm dificuldade de traduzir a solução para um programa de computador equivalente" (p. 17).

VARIÁVEIS PSICOLÓGICAS NA APRENDIZAGEM DA PROGRAMAÇÃO

Os estudos mostram que características como prévio conhecimento matemático e de ciências têm apresentado correlação com o desempenho dos alunos (Byrne & Lyons, 2001; Chumra, 1998; Wilson & Shrock, 2001). No entanto, acredita-se que esta correlação está mais ligada aos processos mentais que os alunos devem adquirir a fim de resolver problemas matemáticos e de laboratório, que são os mesmos ou similares aos necessários para a programação. Surpreendentemente, estudos apresentados por Bennedsen e Caspersen (2006) concluem que não existe correlação entre capacidade de abstração e aprendizagem de programação. Também a experiência prévia em computação não é um factor determinante, mesmo podendo contribuir para melhores resultados em cursos introdutórios de programação (Allert, 2004; Byrne & Lyons, 2001). Por outro lado, características demográficas (etnia, idade, género) dos alunos não se apresentam correlacionadas com o sucesso em programação (Evans & Simkin, 1989).

Os estilos de aprendizagem dos alunos também têm sido considerados nestes estudos. Entre as provas mais usadas encontra-se o *Kolb's Learning Style Inventory* (Kolb, 1985) e o *Felder-Silverman Index of Learning Styles* (Felder & Silverman, 1988). O *Kolb's Learning Style Inventory* tem como base a teoria da aprendizagem pela experiência, um ciclo de quatro estágios onde a *experiência concreta (CE)*, serve como base para a *observação reflexiva (RO)*, que leva a uma *conceptualização abstrata* (AC), que guia a *experimentação activa (AE)*. Cada indivíduo tem uma abordagem preferida e que se pode traduzir num dos quatro estilos de aprendizagem: convergente, divergente, assimilador ou acomodador. Um outro questionário bastante usado é o *Felder-Silverman Index of Learning Styles*, desenvolvido para a área de engenharia e recorrendo a quatro dimensões para descrever as formas de aprender. Na primeira dimensão, os sujeitos podem ser classificados como aprendizes *activos*, que aprendem através da experimentação e trabalhando com outros, ou *reflexivos*, que aprendem pensando ou reflectindo sobre as coisas, geralmente sozinhos. A segunda dimensão classifica os aprendizes como *sensitivos* ou *intuitivos*, onde os primeiros são concretos, práticos e orientados a factos e processos, enquanto os outros são conceituais, inovadores, e orientados a teorias e significados. Na terceira dimensão, os sujeitos são classificados como aprendizes *visuais*, que preferem figuras, diagramas, e esquemas, ou

verbais, que preferem explicações escritas ou orais. A última dimensão classifica os sujeitos como aprendizes *sequenciais*, que aprendem de maneira incremental, ou *globais*, que aprendem em grandes saltos.

A influência do estilo de aprendizagem do aluno nos resultados de cursos introdutórios de programação tem sido ambígua. O estudo conduzido por Byrne e Lyons (2001), usando o *Kolb Learning Style Inventory*, concluiu que não existe correlação entre o estilo de aprendizagem e o desempenho, apesar de 37% dos alunos da turma serem convergentes e terem obtido melhores resultados. Pillay e Jugoo (2005) realizaram dois estudos, utilizando também o *Kolb Learning Style Inventory*; num estudo concluíram que assimiladores têm melhor resultado que divergentes; o outro estudo foi inconclusivo. Por sua vez, Allert (2004) e Thomas, Ratcliffe, Woodbury e Jarman (2002), ambos utilizando o *Felder-Silverman Inventory*, reportam que alunos reflexivos e verbais têm melhor resultado em cursos de programação do que alunos activos e visuais. Estes resultados alinham-se com as observações de Felder de que o ensino de engenharias tende para os estilos de aprendizagem reflexivo, intuitivo, verbal e sequencial.

Para além dos estilos de aprendizagem, outras variáveis cognitivas aparecem consideradas na investigação nesta área. Uma delas passa pelos esquemas ou modelos mentais. A importância de modelos mentais para a compreensão de programas já foi largamente estudada (Cañas, Bajo, & Gonzalvo, 1994; Pennington, 1987; Soloway & Ehrlich, 1984; Wiedenbeck, LaBelle, & Kain, 2004). A importância atribuída aos modelos mentais parece estar associada à clareza conceptual. Neste sentido, acredita-se que a existência modelos mentais adequados é fundamental na elaboração de programas, que são uma representação directa dos primeiros. No seu artigo "Programming as Theory Building", Naur (1985) discorre sobre a necessidade de o programador ter uma "teoria" clara sobre como o programa resolve o problema que está a ser tratado. A qualidade do programa fica assim directamente determinada pela relação existente entre a qualidade da teoria sobre o problema e a teoria da solução.

Os modelos mentais têm sido estudados em relação a diferentes aspectos de programação e usados de diferentes maneiras para avaliar alunos de computação. Vários autores analisaram o modelo mental de alunos com relação ao funcionamento do computador (Mayer, 1989; Perkins, Schwartz, & Simmons, 1988). Segundo Mayer, alunos que não tinham um modelo mental correcto de como o computador manipula variáveis e dados na memória têm maior dificuldade em entender os comandos das linguagens de computador. Cañas, Bajo e Gonzalvo (1994) verificaram que sujeitos têm diferentes representações mentais de programas de computação, as quais podem ser baseadas em aspectos sintácticos ou em aspectos semânticos, aproximando-se mais do modelo mental encontrado em *experts*. Segundo Denhadi (2006), os alunos que obtêm sucesso em cursos de programação são aqueles que usam um modelo mental, seja ele qual for, para a atribuição e manipulação de variáveis de maneira consistente.

Outros autores têm-se preocupado com as diferenças de modelos mentais entre *experts* e noviços. Enquanto os noviços vêm programas como sequências de comandos, os *experts* agrupam comandos em esquemas que representam funcionalidades. Nestes casos, pode-se analisar a representação ou modelo mental de um sujeito através de tarefas como *recall*, categorização ou associação por semelhança, que parecem ser sensíveis a mudanças que ocorrem na representação mental à medida que os alunos aprendem a programar (Ackermann & Stelovsky, 1987). Ao contrário das expectativas originais, o modelo mental dos alunos não é afectado por experiências anteriores de programação no ensino secundário. No entan-

to, desenvolver modelos mentais leva a maior auto-eficácia e a melhores resultados, podendo representar 30% da variância nas classificações (Wiedenbeck, LaBelle, & Kain, 2004).

Este artigo, auscultando professores e estudantes, pretende perceber as razões subjacentes às dificuldades sentidas por alguns na aprendizagem da programação. Neste sentido procura-se contrastar as funções cognitivas que poderão explicar a diferenciação entre um bom e um fraco desempenho académico nas disciplinas de programação, postulando de seguida algumas formas de intervenção na área.

METODOLOGIA

PARTICIPANTES

Participaram neste estudo professores e alunos universitários da disciplina de Introdução à Programação, provenientes da Universidade Federal de Goiás (Brasil) e da Universidade do Minho (Portugal). Procurou-se que no grupo de estudantes estivessem representados sujeitos com facilidade e sujeitos com dificuldades na aprendizagem da programação não obstante a aplicação de esforço, para maior contrastação das suas respostas na entrevista. No total, foram entrevistados cinco alunos com dificuldades e quatro alunos com um bom desempenho. Dos nove alunos, apenas um era do sexo feminino. A grande maioria dos alunos estava a frequentar a disciplina pela primeira vez, sendo que apenas um era repetente; dois dos alunos já tinham tido algum contacto prévio com programação antes do curso. Em relação aos professores, foram entrevistados três professores portugueses e dois professores brasileiros.

INSTRUMENTOS

O presente estudo teve por base uma entrevista semi-estruturada a alunos do Brasil, e a professores do Brasil e de Portugal. Nestas entrevistas procurou-se identificar a que factores se atribuem as dificuldades dos alunos na aprendizagem de programação. Ao invés de questões motivacionais ou referentes ao estilo de aprendizagem e método de trabalho, comuns a muitos outros alunos dos mais diferentes cursos, procurou-se um enfoque nas variáveis cognitivas, ou seja, possíveis funções ou habilidades cognitivas a que professores e alunos atribuem a facilidade ou a dificuldade na aprendizagem da programação. No caso concreto dos professores, a entrevista centrou-se na sua experiência e explicações pessoais para a situação de alunos que aprendem com muita facilidade ou com muita dificuldade a programar. Também se procurou que, na base da sua experiência, tomassem alunos que sentiram muitas dificuldades mas que, num dado momento, acabaram por superá-las e conseguiram programar.

PROCEDIMENTOS

A selecção dos sujeitos entrevistados foi feita pelos professores com base nas classificações dos alunos e na observação levada a cabo durante as aulas. Os alunos foram informados dos objectivos do estudo a acederam livremente a participar. As entrevistas foram colectivas, ou seja, uma com o grupo de bons alunos e outra com alunos com dificuldades na aprendizagem. A entrevista foi dirigida por dois psicólogos e um professor de programação, havendo a preocupação de focar a reflexão dos alunos nas funções cognitivas da aprendizagem dos conteúdos curriculares da programação e de estimular a participação de todos os alunos do grupo. Em situações pouco claras do discurso, procurou-se que o aluno exemplificasse a situação, ou então, apelava-se a um outro aluno do grupo que descrevesse por suas palavras a ideia que estava a ser transmitida.

RESULTADOS

Tomando as respostas obtidas nas entrevistas a professores e a alunos, privilegiaremos os conteúdos reportados às funções cognitivas

inerentes à aprendizagem da programação. Assim, questões motivacionais, processos de ensino durante a frequência do ensino secundário, capacitação pedagógica do professor, organização das instituições e dos cursos, entre outras dimensões possíveis, não vão ser aqui consideradas em virtude do enfoque deste estudo.

Segundo os professores, é clara a distinção entre um grupo de alunos que aprende facilmente e avança sem dificuldades intransponíveis na disciplina e um outro grupo que experiencia imensas dificuldades, por vezes acabando por desistir do curso ou realizando a disciplina de programação com fraca classificação. Da mesma forma, identificam um terceiro grupo que inicialmente experiencia muitas dificuldades mas que, num dado momento da sua aprendizagem, ultrapassa esse impedimento e começa a compreender as tarefas de programação específicas do momento da aprendizagem em que se encontra. Em relação a este fenómeno, considera-se que os alunos, anteriormente com dificuldades, exibem uma espécie de "salto": vivenciam um *insight* que lhes permite transitar de uma situação em que a sua compreensão enfrentava dificuldades para uma posição em que ficam capacitados para realizar as tarefas de programação.

Na análise destes subgrupos de alunos, os professores parecem acreditar que existem dois grandes factores, ou seja, duas habilidades cognitivas essenciais na melhor ou pior aprendizagem da programação. Uma primeira habilidade cognitiva reporta-se à capacidade de sair de uma análise mais holística dos problemas do quotidiano e fazer uma leitura mais analítica dos mesmos, realizando inclusive um planeamento sequencial. Quase em termos de um "estilo cognitivo", o aluno terá de sair do global e do óbvio, e passar para uma análise mais minuciosa, em que cada solução é planificada passo-a-passo (mesmo que no seu quotidiano não seja essa a forma mais usual de perceber e resolver os problemas). Em suma, os alunos precisam de uma visão global do problema e da sua solução, mas é igualmente essencial uma visão das partes que irão compor a solução e que devem ser organizadas e executadas de maneira sequencial. Em sua opinião será esta transição, permitindo "ver o todo a partir das partes", que permitirá ao aluno elaborar o algoritmo. Nas palavras dos professores, pensando desta forma, o aluno deixa de "pensar como pessoa" e passa a "pensar como um computador", levando em conta os processos limitados de compreensão por esta máquina. Acrescentam que "pensar como um computador" é, com frequência, apelidado na área da informática como "raciocínio algorítmico".

Uma segunda habilidade cognitiva mencionada pelos professores é a capacidade de abstracção. Por seu intermédio, espera-se que os alunos passem do mundo concreto para uma outra realidade mais semântica e simbólica, formulando ou representando um problema de forma mais abstracta e menos presa aos pormenores e elementos singulares concretos. Quando um aluno tem dificuldades em fazer tal abstracção, ele não transita da realidade palpável para uma linguagem mais genérica, como por exemplo a notação exigida na aprendizagem da programação.

A apetência para proceder à abstracção está ligada à necessidade de gerar soluções genéricas, aplicáveis a várias situações. Assim, ao tentar encontrar uma solução para um dado problema, o aluno não fica ligado a dados específicos; pelo contrário, busca abstrair da solução o "processo", passível de ser futuramente aplicado em várias instâncias. Por exemplo, supondo que o problema é encontrar a hipotenusa de um triângulo, o aluno tem que encontrar uma solução que permita encontrar a resposta para quaisquer lados fornecidos, e não para um triângulo específico.

Para além das habilidades cognitivas aqui tratadas, existirão outras variáveis implicadas.

A título de exemplo, alguns professores acreditam que a habilidade matemática é importante para o sucesso na programação; todavia, a maioria acredita que a correlação de sucesso em ambas as disciplinas está ligada a funções cognitivas mais básicas que seriam comuns às duas áreas. Assim, ao conseguir desenvolver uma "lógica matemática", o aluno adquire estruturas ou habilidades cognitivas que facilitam ou promovem a aprendizagem da programação.

Tomando as respostas dos estudantes nas entrevistas, e em primeiro lugar o discurso dos alunos com fraco rendimento na programação, importa referir que este subgrupo de alunos parece transitar automaticamente da leitura de um problema para o código, negligenciando o momento intermédio da elaboração do algoritmo. Se assumirmos na resolução de um problema três fases (input, processamento e output), estes alunos parecem "saltar" da primeira para a terceira fase sem o necessário trabalho de processamento da informação codificada e objectivo da tarefa. Quanto às dificuldades encontradas nos três momentos, no que concerne a primeira etapa, estes alunos acreditam que a leitura do problema é feita sem obstáculos, sendo que as dificuldades porventura encontradas têm origem em ambiguidades da língua portuguesa. Quanto à etapa final, concordam que a linguagem de programação é difícil, destacando aspectos específicos como a passagem de parâmetros e o uso de funções. Relativamente à etapa intermédia, e quando questionados sobre a necessidade de alguns procedimentos intermédios entre a leitura do problema e o ensaio da solução, a elaboração do algoritmo não parece constituir-se como um entrave. No entanto, estes alunos sentem particulares dificuldades quando tentam encontrar uma solução abstracta para o problema em mãos. Tomando as suas palavras, "a dificuldade está na compreensão, na transição da linguagem para o código. No raciocínio que é necessário efectuar".

Estes alunos acham que o programa deve ser, de início e de forma completa, estruturado mentalmente "na cabeça", e apenas depois concretizado. Todavia, saltam esta tarefa à frente, exibindo uma tendência para começar a programar de imediato, antes de terem a solução completa do problema delineada. No fundo, estes alunos contentam-se em ter uma ideia inicial e logo passam a implementá-la, por tentativa e erro, sem antes construírem uma visão global da solução. Por outro lado, quando questionados sobre que aspectos apresentam maior complexidade na elaboração da solução abstracta, não conseguem elaborar as suas respostas; sentem a dificuldade, mas não sabem exactamente porquê ou em qual aspecto. Na verdade, não se deterão muito tempo nesta questão, daí que a negligenciem no decorrer da programação e não a tivessem mencionado espontaneamente quando entrevistados.

Importa agora cruzar a percepção dos alunos com dificuldades com a opinião dos professores, anteriormente apresentada. Quanto à capacidade de "pensar como um computador", i.e., de fazer uma leitura analítica e um planeamento sequencial dos problemas diários, que possibilitam a elaboração do algoritmo, esta não parece estar consolidada nos alunos. Apenas um deles disse recorrer à subdivisão do problema em problemas menores para obter a solução: segundo o mesmo, o problema é como um quebra-cabeças, que se vai juntando pecinha a pecinha, de modo a montar o produto final. A divisão facilita o processo de resolução de um problema: ao dividir o problema, é mais fácil fazer um algoritmo para cada um dos problemas menores; faz-se um algoritmo para cada uma das partes e depois estas são ligadas. É necessário dividir em funções, de modo a ter uma maior organização e, assim, solucionar o problema, pois "Aí é possível identificar o erro, formalizá-lo e consertá-lo." É interessante ressaltar que esta metodologia já era adoptada pelo aluno para a compreensão de qualquer matéria anteriormente à sua entra-

da no curso, o facto de trabalhar com programação tornou esta rotina mais forte.

De uma forma geral, estes alunos parecem "queimar etapas" – neste caso, a da semântica, de elaboração do algoritmo. Avaliam como mais fácil ler um programa já escrito, ou então, transitar reactivamente da leitura de um problema inicial para a escrita de uma solução/código. Por outro lado, apresenta-se mais intrincado fazer uma leitura pessoal da solução, extraindo dessa interpretação um algoritmo: "A leitura está mais "fora" da realidade, não é próxima dela, não é natural.". Com efeito, para fazer essa leitura torna-se necessário "pensar como um computador", o que se afigura para eles complexo.

Quanto à habilidade cognitiva de abstracção, quer professores quer alunos a identificam como pertinente. Ainda que estes alunos não sintam que a programação ajude na resolução de problemas no mundo real, acreditam que a resolução de problemas é facilitada pelo processo de imaginar como funciona concretamente na vida real – fazendo-se uma abstracção. No que concerne o processo de programação em traços gerais, os alunos destacam a capacidade de raciocínio e algum nível de criatividade, avaliada por eles como um simples adereço (interface). A criatividade é apenas mais forte de início, na interpretação do aspecto formal; depois, os programas em si são apenas raciocínio. Por exemplo, ao programar jogos de computador, o raciocínio está envolvido na estruturação do jogo, enquanto que a criatividade está implicada na criação de condições para captar e manter a atenção no jogo.

Na entrevista aos alunos com bom desempenho, verifica-se a importância atribuída à fase intermédia de pensar previamente a solução, acreditando ser fundamental. De forma geral, destacaram a importância de compreender a lógica do problema e, para extrair essa lógica, deve-se analisar o problema de forma geral e procurar o seu foco: "Primeiro é necessário compreender o problema, procurar soluções, pensar de formas diferentes e escolher a solução que se considera melhor". Quando indagados sobre o momento mais desafiante da programação, este também parece referir-se à etapa da elaboração do algoritmo, o que implica fazer uma abstracção e identificar a lógica de um problema. Estes alunos consideram que a dificuldade reside no raciocínio para a generalização de soluções. Comentam que há colegas que nem tentam chegar a uma solução generalizável: "Não conseguem percepcionar um problema como um todo e dividi-lo em partes. Para isto, é necessário ter mais experiência, "malícia", que é a parte mais difícil". Talvez assim se explique por que consideram mais difícil ler programas elaborados por outra pessoa do que escrever programas. A primeira é mais desafiante, pois obriga a compreender a lógica que orientou o pensamento de outra pessoa – "É a lógica ao contrário" –, permitindo adquirir mais competências.

Convém agora articular a percepção destes alunos com o discurso dos professores. No que concerne a capacidade de leitura analítica e planeamento sequencial dos problemas, antecedendo a elaboração do algoritmo, esta parece ser igualmente valorizada por professores e alunos. Um dos alunos afirmou que costuma simplesmente "atacar" o problema como um todo, mas, face a dificuldades, opta por dividir o mesmo em partes menores. Outro afirma que a lógica é "subtil de pegar", mas que determinadas ferramentas visuais podem ser de ajuda, como é o caso dos fluxogramas: é necessário "colocar o problema no papel. Só pela cabeça não é possível". Com efeito, quando face a dificuldades, parece ser vantajoso esquematizar o problema, procedendo à sua divisão em tópicos e à análise da forma como se relacionam, criando uma estratégia para cada parte: "É esta a chave para pegar num problema complexo. Dividi-lo e ver como as partes se relacionam entre si". Procura-se executar algumas partes (menores) do algoritmo na cabeça, não todas, pois é demasiado difícil. Para os alunos, além

da lógica, é necessária uma prática ou habilidade adquirida para compreender como funciona o algoritmo, como forma de se visualizar a solução. No que respeita a habilidade cognitiva de abstracção, estes alunos também lhe dedicam importância.

Ponderando o processo de programação de uma forma geral, os alunos consideram que a programação se baseia fundamentalmente em raciocínio, embora também se sirva de forma significativa da criatividade. A título de exemplo, os alunos indicam que programar implica a visualização de uma solução, requer antecipar a lógica inerente a um problema. "Achar a solução é raciocínio. Refinar a solução é criatividade". Para além disso, estão envolvidas boas capacidades matemáticas, ou então, sentir gosto pela matemática, pois facilita o raciocínio e é a base para conseguir fazer um algoritmo.

DISCUSSÃO E CONCLUSÃO

Nas disciplinas introdutórias de programação de computadores, tem-se verificado internacionalmente que uma parte dos alunos, mesmo demonstrando interesse e empenho na aprendizagem, não consegue obter sucesso. Outros, no entanto, não obstante as suas dificuldades, conseguem a dado momento dar um "salto" que lhes permite compreender a disciplina e avançar. Para tentar isolar os factores que explicam tais dificuldades ou que permitem este "salto", procurou-se identificar, através da análise da percepção do problema pelos actores envolvidos, as variáveis que parecem influenciar o baixo desempenho dos alunos.

Professores e alunos com bom desempenho apontam a fase de desenvolvimento do algoritmo como crítica. Basicamente, dois processos cognitivos foram identificados como sendo responsáveis pelas dificuldades reveladas pelos alunos: a capacidade de organizar a solução em passos sequenciais, i.e., de elaborar um algoritmo para a solução, e a capacidade de abstracção, entendida como a capacidade de encontrar soluções genéricas para os problemas, que pode eventualmente envolver identificar padrões para problemas aparentemente diferentes. Para facilitar estes processos, os alunos aplicam muitas vezes a estratégia de decompor o problema em subproblemas, identificando as relações entre as partes. Importa ressalvar que esta estratégia é apontada como uma boa prática no desenvolvimento de programas, sendo que existem estruturas nas linguagens de programação especialmente adequadas a implementar esta estratégia.

Enquanto que os alunos com um bom desempenho tentam obter uma maior compreensão do problema e uma solução global para o mesmo antes de começar a programar, alunos com um baixo desempenho descartam a parte de pensar numa solução abstracta, passando logo para a programação. Pode-se entender que, na primeira abordagem, os alunos formam uma teoria do problema e da solução, segundo a definição de Naur (1985), o que contribui para a elaboração de melhores programas. Podemos, nestas situações apelar a presença ou ausência de estratégias de autoregulação por parte dos alunos, nomeadamente na compreensão da informação necessária à resolução dos problemas (Solano-Pizarro, González-Pienda, González-Pumariega, & Núñez-Pérez, 2004). Neste sentido, a programação passa a ser uma tarefa, onde o algoritmo da solução é mapeado de forma directa para a linguagem de programação. Esta diferença de estratégias que pode traduzir a presença ou ausência de capacidades cognitivas subjacentes, pode ser responsável pela percepção distinta da dificuldade atribuída às diferentes fases de programação. Enquanto alunos com bom desempenho avaliam como mais difíceis as fases iniciais, ligadas à elaboração de uma solução abstracta para o problema, os alunos com um baixo desempenho atribuem maior dificuldade à fase prática de implementação da solução, ou seja, de programação.

Todavia, estas diferenças não justificam a ocorrência do "salto" que ocorre em alguns alunos. Acreditamos que este pode estar ligado à existência de determinados modelos mentais, favorecedores da aprendizagem da programação. Por um lado, pode inferir-se que aqueles alunos que iniciam a disciplina com modelos mentais que auxiliam na resolução de problemas têm mais facilidade na mesma. Estes modelos podem ter sido adquiridos em disciplinas de matemática ou ciências, o que explicaria o facto dos alunos com bom desempenho nessas disciplinas tendencialmente apresentarem um bom desempenho em disciplinas de programação. Por outro lado, no que respeita os alunos que eventualmente produzem o "salto", esses possuirão um esquema de representação da realidade minimamente propício à aquisição das tarefas de programação; detêm, assim, uma base que permite encaixar os outros conceitos, a partir da qual ocorre o "salto", que lhes permite uma melhor compreensão e desempenho na disciplina.

Vale observar que na literatura, *experts* e noviços são apresentados como estruturando o seu conhecimento de maneira distinta: enquanto os *experts* se focam na semântica dos comandos, os noviços focam-se na sintaxe (Cañas, Bajo, & Gonzalvo, 1994; Pea & Kurland, 1984). Esta questão foi, de facto, verificada nas entrevistas realizadas: os alunos com melhor desempenho demonstraram colocar maior importância nos aspectos semânticos dos problemas, dedicando-se à compreensão profunda dos mesmos e do funcionamento da solução. Ao focar aspectos semânticos, eles podem estar a desenvolver modelos mentais mais próximos dos usados por *experts*. Pelo contrário, os alunos com dificuldades centram-se na sintaxe, i.e., numa leitura primária e superficial do problema, não alcançando o cerne do mesmo. Isto dificultará a criação de modelos mentais adequados para a aprendizagem eficiente da programação.

Neste sentido, as dificuldades apresentadas pelos alunos podem estar ligadas a défices que se arrastam do ensino secundário. A questão não está essencialmente ao nível dos conteúdos curriculares aprendidos mas de um conjunto de habilidades e estratégias associadas à melhor ou mais fraca aprendizagem, interessando por isso uma análise sistémica do problema (Peralbo-Uzquiano & Barca-Lozano, 2003). A título de exemplo, na prova do PISA 2003, nas questões relacionadas com a resolução de problemas, os processos avaliados foram muito próximos dos necessários em programação, verificando-se que menos de 20% dos alunos conseguem resolver problemas difíceis (Nivel 3). Nestes, o aluno deve ser capaz de analisar uma situação e tomar decisões, manipular múltiplas condições simultaneamente, pensar sobre as relações subjacentes a um problema, resolvê-lo de maneira sistemática, avaliar o seu trabalho e comunicar os resultados. No Brasil, mais de 50% dos alunos testados não conseguiram resolver problemas do Nivel 1 (mais fáceis). Talvez não por coincidência, tiveram também um pobre desempenho na avaliação de leitura. Em Portugal, 25% não atingiu o Nível 1.

Um outro aspecto interessante que se pode comunicar a partir da análise dos dados do PISA 2003 é que existe uma elevada correlação entre resolução de problemas e outras áreas de actuação, especialmente a matemática. No entanto, a *performance* na resolução de problemas matemáticos que envolvem simples cálculos, sem maiores inferências, apresenta baixa correlação com a *performance* na resolução de problemas. Estas conclusões alinham-se com a avaliação dos professores de que o conhecimento matemático não determina o sucesso na programação; estes acreditam que a correlação entre matemática e programação está ligada ao desenvolvimento de processos mentais comuns às duas áreas, como por exemplo a inferência.

Para além dos modelos mentais, e embora o âmbito deste estudo não fossem os estilos de

aprendizagem, a verdade é que estes poderão estar relacionados com o "salto" aqui explanado. Como definido na última dimensão de Felder e Silverman (1988) quando se referindo aos diferentes estilos de aprendizagem dos alunos – sequenciais ou globais –, alguns alunos aprendem de forma contínua enquanto outros aprendem por "saltos". Assim, os "saltos" percebidos nos alunos de programação podem estar ligados ao aluno com um estilo de aprendizagem global. Importa salientar que, nos estudos que utilizaram o inventário de Felder-Silverman, a última dimensão não aparece correlacionada com o aproveitamento na disciplina. Todavia, poderia explicar o facto de que, enquanto o desenvolvimento de alguns alunos é percebido como contínuo – é o caso dos alunos com um estilo de aprendizagem sequencial –, no caso dos alunos com um estilo de aprendizagem global, a sua aprendizagem é verificada a partir de um determinado "salto" dado no seu processo normal de aprendizagem.

Estas possibilidades aqui apresentadas devem ser aprofundadas através de estudos mais sistemáticos, como forma de permitir uma avaliação fundamentada e a identificação de estratégias de ensino mais adequadas ao eficiente processo de ensino-aprendizagem de programação.

REFERÊNCIAS BIBLIOGRÁFICAS

Ackermann, D., & Stelovsky, J. (1987). The role of mental models in programming: From experiments to requirements for an interactive system. *Lecture Notes in Computer Science, 282*, 53-69.

Allert, J. (2004). Learning style and factors contributing to success in an introductory computer science course. In Looi, C-K, Sutinen, E., Sampson, D. G., Aedo, I., Uden, L., & Kähkönen, E. (Eds.), *Proceedings of 4th IEEE International Conference on Advanced Learning Technologies*. Joensuu, Finland, 385-389.

Almeida, L. S. (2007). Transição, adaptação académica e êxito escolar no ensino superior. *Revista Galego-Portuguesa de Psicoloxía e Educación, 15*, 203-215.

Bennedsen, J., & Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bulletin, 38* (2), 39-43.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. In Fincher, S., Klein, B., Culwin, F., & McCracken, M. (Eds.), *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*. NY: ACM, 49-52.

Cañas, J. J., Bajo, M. T., & Gonzalvo, P. (1994). Mental models and computer programming. *International Journal of Human-Computer Studies, 40* (5), 795-811.

Chumra G. A. (1998). What abilities are necessary for success in computer science? *SIGCSE Bulletin, 30* (4), *55a-58a.*

Dehnadi, S. (2006). Testing programming aptitude. In P. Romero, J. Good, E. A. Chaparro, & S. Bryant (Eds.), *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group*. University of Sussex, 22-37.

Evans, G. E., & Simkin, M. G. (1989). What best predicts computer proficiency? *Communications ACM, 32* (11), *1322-1327.*

Felder, R. M., & Silverman, L. K. (1988). Learning and teaching styles in engineering education. *Engineering Education, 78* (7), 674-681.

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. Loughborough: University United Kingdom, 53-58.

Kolb, D. A. (1985). *Learning style inventory*. Boston, MA: McBer and Company.

Leither, H. E., & Lewis, H. R. (1978). Why Johnny can't program: A progress report. *SIGCSE Bulletin, 10* (1), 266-276.

Mayer, R.E. (1989). The psychology of how novices learn computer programming. In E. Soloway, & J.C. Spohrer (Eds.), *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Naur, P. (1985). Programming as theory building. *Journal of Systems Architecture: The Euromicro Journal*, *15* (5), 253-267.

OCDE (2003), PISA 2003: First Results from Pisa 2003 - Executive Summary. Disponível em http://www.oecd.org/dataoecd/1/63/34002454.pdf. Acedido em 12 Novembro 2010.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive prerequisites of learning computer programming (Technical Report No.18). New York: Bank Street College of Education, Center for Children and Technology.

Pennington, N. (1987). Comprehension strategies in programming. In E. Soloway, & S. Iyengar (Eds.), *Empirical studies of programmers: Second workshop* (pp. 100-113). Norwood, NJ: Ablex.

Peralbo-Uzquiano, M., & Barca-Lozano, A. (2003). El fracaso escolar ?Cómo argumento?. *Revista Galego-Portuguesa de Psicoloxía e Educación, 9*, 167-182.

Perkins, D. N., Schwartz, S., & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 153-178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pillay, N., & Jugoo, V. (2005). An investigation into student characteristics affecting novice programming performance. *SIGCSE Bulletin*, *37* (4), 107-110.

Solano-Pizarro, P., González-Pienda, J. A., González-Pumariega, S., & Núñez-Pérez, J. C. (2004). Autorregulacón del aprendizaje a partir de textos. *Revista Galego-Portuguesa de Psicoloxía e Educación, 9*, 111-128.

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programmer knowledge. *IEEE Transactions of Software Engineering, SE-10* (5), 595-609.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *SIGCSE Bulletin, 34* (1), 33-37.

Wiedenbeck, S., LaBelle, D., & Kain, V. (2004). Factors affecting course outcomes in introductory programming. In E. Dunican, & T. R. Green (Eds.), *Proceedings of the 16th Workshop of the Psychology of Programming Interest Group*. Carlow, Ireland: Institute of Technology Carlow, 97-110.

Wilson, B. C., & Shrock, S. (2006). Contributing to success in an introductory computer science course: A study of twelve factors. *ACM SIGCSE Bulletin, 33* (1), 184-188.

Winslow, L. E. (1996). Programming pedagogy: A psychological overview. *ACM SIGCSE Bulletin, 28* (3), 17-25.